Universidad Carlos III de Madrid
Escuela politécnica superior

INGENIERÍA EN INFORMÁTICA
PROYECTO FIN DE CARRERA

Sistema de indexación de C# .Net para implantar procesos de reutilización UML

Tutor: Juan Llorens Morillo
Autor: Luis Mansilla García

"La inteligencia me persigue, pero yo soy más rápido"

Groucho Marx

Agradecimientos

A mis padres, todo lo que soy y todo lo que pueda llegar a ser se lo debo a ellos. Gracias por vuestra total dedicación y paciencia, por vuestra confianza y respeto, por vuestra compresión y afecto, y en definitiva por hacer de vuestros dos hijos lo más valioso de vuestras vidas y lo que es más importante, por hacernos sentir así a nosotros.

A mi hermana, a la que aunque nunca se lo haya dicho, ni se lo vaya a decir, quiero y admiro como ella no se imagina. Ha sido ella una de las principales razones que me han hecho terminar por fin…tenía que seguir siendo el favorito y no podía ser que acabaras tú la carrera antes que yo.

A mis dos vecinos de Godbyvägen, Nacho y Juan, que hicieron de mi estancia en Mariehamn una de la experiencias más bonitas de toda mi vida, estoy seguro que con ellos dos un Erasmus en Valdemoro hubiera sido también inolvidable (pero con menos vackra flickor). Gracias por vuestra amistad.

A todas aquellas personas que he conocido a lo largo de mis cinco años en la Carlos III. Gracias Rubén por toda tu ayuda, sin ésta seguramente todavía estaría cursando ADA, gracias Arturo, Pablos, Adrian, Samuel, Toni… por hacer de mi paso por la universidad, aunque no conteste los dichosos mensajes del Facebook, algo que recordaré siempre.

A José, David, Víctor, Erika, Sofía, Lisa… y muchos más que seguro que me dejo y que me enseñaron que el mundo es mucho más grande que Leganés. Por vuestra culpa he tardado tanto en presentar el proyecto, tanta fiesta tanta fiesta, jamás os lo perdonaré.

A Juan Llorens por su enorme paciencia y por brindarme la oportunidad de descubrir una ciudad y un país espectacular en el que he conocido a tanta y tanta gente maravillosa.

A todos los trabajadores de la Universidad: profesores, conserjes, becarios, camareras… que han hecho y hacen posible que infinidad de jóvenes disfrutemos de una de las experiencias más enriquecedoras e influyentes de nuestra vida.

Muy especialmente a los que hicieron de la Carlos III una universidad tan exigente y con normas tan severas…gracias de verdad.

Por último, pero por importancia los primeros, a mis abuelos Ramón y Vicenta. Me gustaría ser capaz de expresar con palabra lo que siento por vosotros y lo que habéis y siempre significaréis para mí. ! Abuelo lo conseguí ¡

Introducción

Fundamentos Teóricos

Siglo XXI. Hoy en día es imposible imaginarse el mundo sin tecnología, está presente en nuestras vidas e interactuamos con ella continuamente tanto personal como profesionalmente. No podemos imaginarnos el día a día sin coches, teléfonos móviles o internet, estamos deseando terminar la jornada laboral para poder disfrutar de un buen partido de nuestra selección en la televisión, mientras nos tomamos unas cervezas que se han mantenido frías gracias a nuestro frigorífico. La tecnología hace nuestra vida más fácil y es la herramienta que nos posibilita encontrar solución a problemas complejos o plantearnos nuevos retos a solucionar que nos permitan avanzar y mejorar. Física, química, mecánica, biología… no sería capaz de enumerar todas aquellas disciplinas o campos de estudio que juntos dan forma a lo que conocemos como tecnología, pero sí que me gustaría destacar una: la informática.

La informática es una de las áreas de conocimiento que mayor crecimiento e importancia ha tenido durante los últimos años. Es indispensable para muchas otras áreas científicas y ha revolucionado nuestra vida personal y por supuesto profesional. No importa cuál sea tu ocupación, seguro que la utilizas constantemente y no te sería posible realizar con éxito tú labor sin ella.

Pero la informática sigue siendo un área de conocimiento enorme formada por la unión de un gran número de disciplinas de entre las que de nuevo, me gustaría destacar una: la ingeniería de software. Copiando la definición de ingeniería de software de cualquier diccionario tenemos que: "*Ingeniería de software es la disciplina o área de la informática que ofrece métodos y técnicas para desarrollar y mantener software de calidad*." Y es dentro de esta disciplina donde se encuadra el concepto más importante y en torno al que gira todo este proyecto*:* ***la reutilización del conocimiento***. La información es el activo más importante de cualquier empresa y la reutilización del conocimiento una práctica indispensable para hacer más competitiva y eficaz cualquier organización

La reutilización de conocimiento es una técnica que nos permite y facilita el desarrollo de nuevas aplicaciones ya que podemos aprovechar desarrollos ya existentes para la generación de nuevos programas y aprovechar desarrollos ya existentes significa mejorar en tiempo y costes de producción, variables fundamentales a la hora de abordar cualquier proyecto.

La reutilización de conocimiento, aplicada dentro de la ingeniería del software, puede estar presente a lo largo de todo el ciclo de vida, desde las fases de mayor abstracción: Análisis de requisitos o especificación a aquellas más concretas: diseño o codificación.

Contexto

Este proyecto ha sido realizado en el seno del grupo "Knowledge Reuse" perteneciente al Departamento de Informática de la Universidad Carlos III de Madrid.

Dentro de la reutilización de conocimiento uno de los proyectos en los que trabaja este departamento es el de la creación de indexadores de conocimiento. Estos indexadores son parte de programas encargados de rastrear la web en busca de posible información susceptible de ser indexada. Básicamente un indexador extrae datos de un determinado medio y transforma estos datos en información que nos permita realizar la reutilización. El medio del que extraer la información puede ser muy variado: código, documentos de especificación, documentos de diseño… y si bien el origen puede ser muy diferente el final debería ser el mismo: un formato común de representación de información que permita generar una base de conocimiento homogénea, UML.

Como se ha indicado anteriormente un indexador puede obtener información de diferentes medios, pero si se busca la máxima eficiencia desde el punto de vista de la reutilización, es decir, que podamos tener un nuestra disposición una gran base de información obtenida por indexación, cuanto más extendidos y más generalizado este el uso de estos medios mejor para el propósito.

Dentro de este grupo de tecnologías de gran expansión dentro del mundo de la informática, y más concretamente dentro del desarrollo software, encontramos el entorno de desarrollo de Microsoft .Net. Microsoft .Net son un conjunto de tecnologías que facilitan la creación de aplicaciones, sistemas web o servicios web siendo además uno de las más utilizadas hoy en día. Es por ello que realizar la indexación de aplicaciones codificadas con este estándar permitirá crear una gran base conocimiento.

Objetivo

El objetivo de este proyecto es la creación de un sistema informático que permita analizar y obtener toda la información de los ficheros de código C Sharp (C#) contenidos dentro de una solución .Net. Una vez analizados e indexadas los ficheros de código el sistema debe ser capaz de, gracias al uso de nUML, generar un fichero xmi "XML Metadata Interchange" que recoja toda esta información y del que se pueda inferir directamente un modelo de representación UML.

Glosario

- **Software**: Se conoce como softwarehttp://es.wikipedia.org/wiki/Software - cite_note-0 al equipamiento lógico o soporte lógico de una computadora digital; comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas.

- **UML**: UML es un popular lenguaje de modelado de sistemas de software. Se trata de un lenguaje gráfico para construir, documentar, visualizar y especificar un sistema de software. Entre otras palabras, UML se utiliza para definir un sistema de software.

- **nUML**: Tecnología que da soporte al metamodelo de UML 2.0 (UML Superstructure 2.0) y patrón facade. Esta implementado en .Net de ahí el comienzo de la yuxtaposición de la n a la palabra UML.

- **Microsoft .Net**: es un framework de Microsoft que hace un énfasis en la transparencia de redes, con independencia de plataforma de hardware y que permita un rápido desarrollo de aplicaciones.

- **C Shar (C#):** Es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO.

- **Microsoft Visual Studio**: Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros. A lo largo de esta memoria es posible que esté referenciado con los acrónimos VS.

- **EnvDTE y EnvDTE80**: son bibliotecas COM que contiene los objetos y miembros para la automatización básica de Visual Studio.

- **API**: Grupo de rutinas (conformando una interfaz) que provee un sistema operativo, una aplicación o una biblioteca, que definen cómo invocar desde un programa un servicio que éstos prestan
- **DLL**: Es la implementación de Microsoft del concepto de bibliotecas (librerías) compartidas en sistemas Windows y OS/2.

- **Pre-modelo**: Soporte de información intermedia utilizada por la aplicación para almacenar la información obtenida de los ficheros .cs analizados. Serán utilizados por la librería nUML para transformar esa información en el correspondiente modelo UML.

- **Xmi**: XMI o XML Metadata Interchange (XML de Intercambio de Metadatos) es una especificación para el Intercambio de Diagramas.

- **Zip**: Formato de compresión de archivos.

Estado del arte

En este apartado se listarán todas aquellas tecnologías relacionada con el proyecto. Se realizara una pequeña descripción que permita entender mejor cual es el estado de estas tecnologías actualmente y se identificara también su influencia con el sistema.

Microsoft .Net

Microsoft .Net es un framework de Microsoft orientado a la creación de aplicaciones, que permite un desarrollo rápido y eficaz a la vez que robusto y que nace como respuesta a otras tecnologías similares como la plataforma Java, de Sun Microsystems o los diversos frameworks de desarrollo web basados en PHP.

.Net Framework reúne un conjunto de lenguajes y servicios que simplifican el desarrollo de aplicaciones permitiendo junto con el entorno de desarrollo Visual Studio, no solo la creación sino también las distribución y ejecución de las aplicaciones desarrolladas en la plataforma Microsoft.

Las dos características más importantes de este entorno de trabajo son:

- *Independencia del lenguaje*: con .Net existe la posibilidad de desarrollar código para la plataforma Windows en una gran variedad de lenguajes. Todos ellos generan al ser compilados un código intermedio conocido como MSIL (Microsoft Intermediate Language).

- *Interoperabilidad:* la plataforma .NET ofrece los mecanismos necesarios para acceder a las funcionalidades implementadas en programas que se encuentran fuera del entorno .NET, lo que amplía considerablemente las posibilidades del desarrollador a la hora de diseñar sus programas.

Los principales componentes del marco de trabajo de .Net son:

- *Conjunto de lenguajes de programación*: Existen más de 30 lenguajes adaptados a .Net, desde los más conocidos como C#, Visual Basic o Visual C++ pasando por otros como Perl o Cobol.
- *La biblioteca de Clases Base o BCL*: Provee los bloques fundamentales para cualquier tipo de aplicación, sea Windows, web o servicio web y maneja la mayoría de las operaciones básicas que se encuentran involucradas en el desarrollo de aplicaciones.

- El entorno común de Ejecución para lenguajes CLR: El CLR es el núcleo del framework de .Net.

El framework de .Net nació alrededor del año 2000 con lo que ha ido evolucionando a lo largo de todo este tiempo hasta hoy. La evolución de las diferentes versiones de .Net así como su relación con las diferentes plataformas tecnológicas y versiones de Visual Studio se puede observar en la siguiente tabla:

Versión .NET	1.0	1.1	2.0	3.0	3.5	4.0
Plataforma	98 a XP	+ Server 2003	+ SQL 2005	XP SP2 Srv 2003 SP1 Vista	Srv 2008 SQL 2008	Win 7
Versión Visual Studio	VS 2002	VS 2003	VS 2005	VS 2005	VS 2008	VS 2010

Influencia en el sistema

El framework de .Net constituye una parte fundamental de la aplicación y ha jugado dos papeles fundamentales y totalmente diferenciados entre sí:

- Ha sido la tecnología de desarrollo utilizada en el proceso creativo del sistema ya que el indexador ha sido codificado utilizando el framework 3.5 y el lenguaje C#.
- Es el objetivo de análisis de la aplicación, ya que son soluciones creadas bajo el estándar .Net y en concreto aquellas en la que se ha utilizado el lenguaje C# las que serán analizadas por el indexador.

Visual Studio

Es un entorno de desarrollo integrado (IDE) para sistemas operativos Windows, que soporta varios de los lenguajes que se han enumerado en el apartado anterior como Visual C++, Visual C#, ASP o visual Basic.

Visual Studio permite crear tanto aplicaciones de escritorio para Windows como aplicaciones o servicios web.

Dado que es el IDE utilizado por los desarrolladores de .Net su evolución ha estado siempre ligada la de este framework, en la tabla anterior se puede observar esta evolución y su relación con los distintos frameworks de .Net.

Influencia en el sistema

Para la consecución del proyecto Visual Studio ha sido una herramienta fundamental que ha participado también con dos roles diferentes:

- Ha sido el IDE utilizado durante todo el proceso de desarrollo.
- Es parte fundamental de la propia ejecución del indexador, ya que la aplicación debe lanzar una instancia de Visual Studio para cargar en ella las solución y poder comenzar el proceso de análisis.

La versión final utilizada para el desarrollo ha sido Visual Studio 2008 aunque se comenzó el desarrollo con la versión anterior Visual Studio 2005.

C Sharp (C#)

Es un lenguaje de programación orientado a objetos, desarrollado y estandarizado por Microsoft como parte de su plataforma .Net.

Comenzó a desarrollarse en el año 1999 por Anders Hejlberg y fue presentado en el año 2000 junto con la plataforma .Net como evolución de C++, pudiéndose definir como un híbrido entre C++ y java. Actualmente se encuentra entre los 10 lenguajes más utilizados.

Las principales características de este lenguaje son:

- **Sencillez**: C# elimina muchos elementos que otros lenguajes incluyen y que son innecesarios en .NET.
- **Autocontenido**: No necesita de ficheros adicionales al propio fuente tales como ficheros de cabecera o ficheros IDL
- **Tamaño de datos fijo**: Independientemente del compilador, sistema operativo o máquina para quienes se compile.
- **Orientación a objetos**: C# soporta todas las características propias del paradigma de programación orientada a objetos: encapsulación, herencia y polimorfismo.
- **Gestión automática de memoria**: Como ya se comento, todo lenguaje de .Net tiene a su disposición el recolector de basura del CLR. Esto tiene el efecto de que no es necesario incluir instrucciones de destrucción de objetos.
- **Modernidad**: C# incorpora en el propio lenguaje elementos que a lo largo de los años ha ido demostrándose son muy útiles para el desarrollo de aplicaciones y que en otros lenguajes como Java o C++ hay que simular.

- **Seguridad de tipos:** C# incluye mecanismos que para asegurar que los accesos a tipos de datos siempre se realicen correctamente, lo que permite evitar que se produzcan errores difíciles de detectar.
- **Sistema de tipos unificado:** A diferencia de C++, en C# todos los tipos de datos que se definan siempre derivarán, aunque sea de manera implícita, de una clase base común llamada *System.Object*, por lo que dispondrán de todos los miembros definidos en ésta clase.
- **Compatible:** Para facilitar la migración de programadores, C# no sólo mantiene una sintaxis muy similar a C, C++ o Java que permite incluir directamente en código escrito en C# fragmentos de código escrito en estos lenguajes, sino que el CLR también ofrece a través de los llamados *Platform Invocation Services (PInvoke),* la posibilidad de acceder a código nativo.

Influencia en el sistema

Al igual que ocurre con la tecnología .Net y Visual Studio, C# ha jugado un rol fundamental en el proceso de creación del sistema y también un doble papel:

- Ha sido el lenguaje de programación utilizado para codificar la aplicación.
- Son los ficheros de código fuente de aplicaciones creadas con C# el objeto de análisis del indexador.

Estándar de compresión .ZIP

Es un formato de almacenamiento sin perdida, muy utilizado para la compresión de datos como imágenes, programas y documentos. El formato ZIP fue creado por Phil Katz y lanzado al público en enero de 1989.

Su principal característica es que comprime cada uno de los ficheros de forma separada lo que permite recuperar los sin tener que leer el resto, aumentando el rendimiento. Sin embargo esta independencia de ficheros provoca también que la ratio compresión sea más bajo que el de otros formatos como 7zip tar.gz, ya que el resultado de agrupar un número grande de pequeños archivos es siempre mayor que agrupar todos los archivos y comprimirlos como si fuera uno solo.

Debido a la gran distribución de .ZIP dentro del mundo informático ha sido este formato el elegido como extensión para los ficheros de entrada que el sistema será capaz de descomprimir y posteriormente analizar.

Influencia en el sistema

Dado que la entrada a del sistema son ficheros con extensión .zip la aplicación ha tenido que implementar una función que permita descomprimir este tipo de archivos con el objetivo de acceder a su contenido y poder comenzar la indexación.

UML

UML es un lenguaje de modelado visual, creado por Rumbaugh, Booch y Jacobson que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Se utiliza para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir. UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema.

UML no es un lenguaje de programación aunque hay herramientas que pueden ofrecer generadores de código de UML para una gran variedad de lenguaje de programación, así como construir modelos por ingeniería inversa a partir de programas existentes.

En 1997 UML 1.1 fue aprobada por la OMG convirtiéndose en la notación estándar de facto para el análisis y el diseño orientado a objetos.

Para este cometido UML presenta distintos tipos de diagramas en los que se pueden apreciar distintos aspectos del sistema. Actualmente la versión más extendida es UML 2.0 en la que los diagramas que existentes son:

- **Diagramas de estructuras:** Este tipo de diagrama muestra los elementos que deben existir en el sistema.
  - Diagrama de Clases.
  - Diagramas de Componentes.
  - Diagrama de Objetos.
  - Diagrama de Estructura Compuesta.
  - Diagrama de Despliegue.
  - Diagrama de Paquetes.

- **Diagramas de Comportamiento:** Muestra lo que debe suceder en el sistema.
  - Diagrama de Actividades.
  - Diagrama de casos de Uso.
  - Diagrama de Estados.

- **Diagramas de Interacción:** Con un cometido similar a los de comportamiento muestran el flujo de control y de los datos entre los elementos que existen en el sistema.

    - Diagrama de Secuencia.
    - Diagrama de Comunicación.
    - Diagrama de Tiempos.
    - Diagrama de vista de Interacción.

Influencia en el sistema

El objetivo final del sistema es el de generar un salida que permita recoger toda la información indexada. Esta salida debe estar representada usando un estándar de manera que se permita homogeneizar la información que los diferentes indexadores serán capaces de obtener. El lenguaje de modelado unificado, UML, es el estándar elegido y por lo tanto la salida final del indexador. Para ser exactos, la salida final del indexador es un fichero de tipo "xmi". Este tipo de ficheros permiten compartir diagramas UML entre diferentes herramientas de diseño lo ofrece la posibilidad de:

- Obtener de forma automática utilizando una gran variedad de herramientas de modelado visual, el modelo UML contenido en el fichero "xmi".
- Al ser "xmi" un estándar permite una fácil distribución de esta información entre diferentes herramientas.

nUML

nUML es una librería para manipular metamodelos descritos en UML 2.0, esta herramienta permite almacenar y manejar los metadatos de los modelos UML. nUML se encuentra implementada en .net, de ahí el origen de su nombre: n (.net)+UML.

Influencia en el sistema

La librería nUML es la herramienta que permite generar el fichero "xmi" con toda la información que el sistema ha sido capaz de indexar. Es por tanto una parte fundamental y de vital importancia para la aplicación ya que cualquier fallo en el uso de esta librería tendría como resultado un modelo UML incorrecto o impreciso.

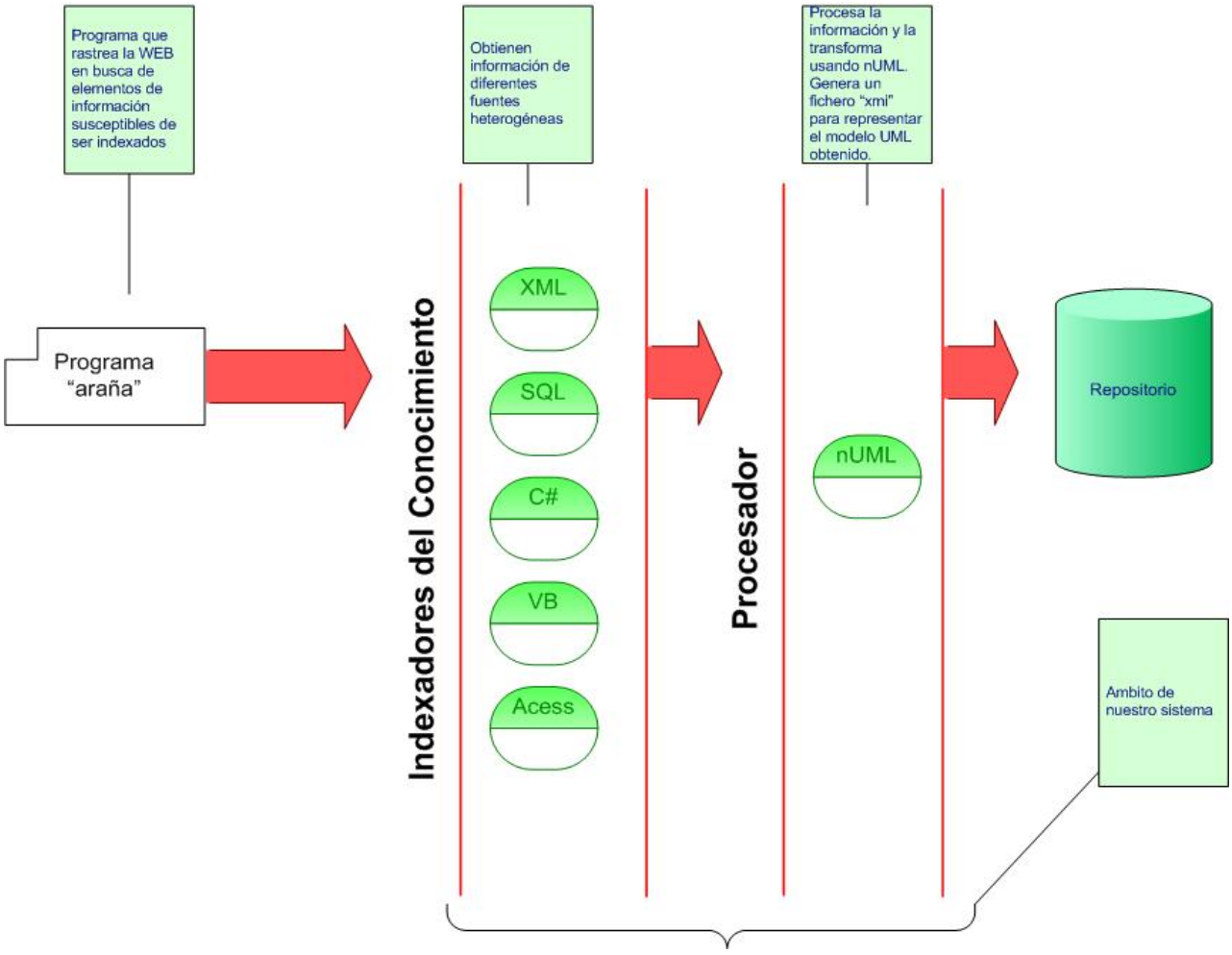
Objetivos, problemas y soluciones

Durante la consecución de este proyecto se ha tenido que abordar una serie de problemas a los que ha sido necesario dar solución y que han constituido un conjunto de hitos remarcables del proceso creativo.

Visión Global

A continuación se dará una visión global del funcionamiento del indexador y su integración dentro de la aplicación matriz. Aunque el siguiente apartado se centrará en los problemas y soluciones que ha sido necesario abordar para la creación del indexador C#, se explicará el contexto general en el que está englobado ya que es de gran ayuda para entender mejor su funcionamiento.

La siguiente figura muestra el entorno de trabajo del indexador C# y su relación con la aplicación matriz.



Se recuerda que el objetivo del proyecto es la creación de un sistema capaz de analizar las soluciones de Visual Studio contenidas en un fichero .zip y generar un modelo UML con la información obtenida de las mismas. En la figura anterior se muestra como queda encuadrado el indexador de C# dentro del sistema matriz.

Se pueden identificar los siguientes elementos:

1. **Programa "araña":** Parte del sistema matriz que se encarga de rastrear la web en busca de soportes de información susceptible de ser indexados. En el caso de este proyecto deberá buscar ficheros .zip. Posteriormente el indexador C# determinará si estos ficheros contienen información válida, es decir soluciones de Visual Studio. El desarrollo de este subsistema queda fuera del ámbito del proyecto.
2. **Indexadores del Conocimiento:** Subsistemas que permiten analizar la información contenida en diferente soportes. Estos subsistemas pueden estar dedicados a una gran variedad de fuentes, como por ejemplo: ficheros xml, sql, acess. En el caso de este proyecto está asociado a los ficheros de código C# contenidos en las soluciones analizadas.
3. **Procesador:** Una vez indexada la información es necesario transformarla en un modelo UML, de manera que a partir de multitud de fuentes heterogéneas: xml, acess, squl, c# obtengamos una salida homogénea UML. De esta transformación se encarga una parte del indexador conocida con el nombre de procesador, que gracias al uso de nUML genera un fichero "xmi" para almacenar el modelo UML.
4. **Repositorio:** Base de datos donde se almacena todo el conocimiento indexado tras este proceso. La creación y gestión de este repositorio queda fuera del ámbito de este proyecto.

A continuación se listarán los problemas y soluciones adoptadas durante la realización del indexador, centrándonos únicamente en los puntos 2 y 3 que son los que abarca este proyecto.

Descompresión de archivos .zip

Objetivo

Realizar la descompresión de ficheros .zip, que son las entradas del sistema, para poder obtener las soluciones contenidas en ellos.

Problema

Como realizar esta descompresión desde código cumpliendo además la premisa de que no sea necesario tener instalado en la máquina host del indexador ningún tipo de aplicación de descompresión.

Solución

Se ha optado por incluir en el proyecto una librería que ofrezca el interfaz necesario para realizar este tipo de operación. De esta forma no es necesario tener instalado ningún tipo de aplicación de descompresión ya que no se tiene la necesidad de acceder ningún API residente en la máquina, pudiendo utilizar directamente desde el código la funcionalidad ofrecida por la dll.

Se realizo una búsqueda por la red de librerías que cumplieran con estos requisitos decidiéndonos finalmente por el uso de la biblioteca IONIC.ZIP por dos razones:

- Cubría totalmente todas las funcionalidades que requeríamos sin la necesidad de recurrir a otras librerías.
- Se trata de una biblioteca de licencia libre.

Creación y manipulación de una instancia de Visual Studio

Objetivo

Crear una instancia de Visual Studio 2008 y cargar en ella las soluciones descomprimidas.

Problemas

Como fase fundamental en el proceso de análisis es necesario cargar en Visual Studio las soluciones .Net descomprimidas. Esto plantea dos problemas:

- Lanzar Visual Studio de forma trasparente al usuario, sin que note la ejecución de la aplicación.
- Abrir la solución deseada en Visual Studio.

Solución

Para ello se ha utilizado las bibliotecas COM EnvDTE y EnvDTE80, que contienen objetos y miembros para la automatización básica de Visual Studio, y que permiten lanzar una instancia de Visual Studio y manipularla. La posibilidad de manipular la instancia creada es fundamental ya que para que la ejecución de Visual Studio sea totalmente transparente en la máquina es necesario suprimir la interfaz de usuario.

Estas mismas bibliotecas ofrecen también la posibilidad de cargar soluciones, utilizando para ello unas serie de rutinas que ponen a nuestra disposición y que serán explicadas con más detalle posteriormente.

Acceso a la información contenida en los ficheros fuente

Objetivo

Extraer toda la información posible de los ficheros de código C# contenidos en las soluciones cargadas.

Problemas

Acceso al código contenido dentro de los ficheros de extensión “cs” e identificación y clasificación de los elementos de información encontrados.

Solución

Para la resolver este problema se ha utilizado de nuevo las librerías EnvDTE y EnvDTE80 que permiten el acceso a los elementos contenidos dentro de una solución de tipo C#, de entre todos ellos a los ficheros de código .cs, y a más bajo nivel permite el análisis e identificación de los elementos de información que componen el código de estos fichero cs.

Generación del modelo UML

Objetivo

Generar, con toda la información recogida de los ficheros “cs”, un fichero “xmi” que permita almacenar toda esta información. De esta forma se recoge toda la información indexada en un modelo UML gracias a la utilización del formato” XML metadata interchange”.

Problema

La correcta utilización de la librería nUML para generar el fichero “xmi” ya que este proceso de traducción es vital para que la salida del sistema sea correcta y precisa.

Cualquier fallo en esta fase provoca que, aunque el proceso de obtención la información se haya realizado correctamente, el modelo de salida que recoge esta información este mal formado y por lo tanto no contenga toda la información deseada.

Solución

Gracias a los ejemplos proporcionados y a la gran ayuda ofrecida por el grupo de “Knowledge Reuse” se ha podido utilizar la librería nUML de forma correcta consiguiendo los resultados deseados.

Metodología de desarrollo

En este punto se analiza la metodología empleada durante el proceso creativo y de implementación de este proyecto.

Este como la mayoría de los proyectos, se basa en la investigación por lo que hasta una vez demostrada su viabilidad y por lo tanto hasta ese punto no se pudo utilizar una metodología clara.

Una vez se encontraron los mecanismos que permitían el desarrollo de la solución para el problema propuesto, es decir, una vez vencida la tecnología se ha aplicado una metodología clásica en cascada, que se ha ido transformando a lo largo del desarrollo en una metodología de cascada incremental, es decir se hace un desarrollo clásico en cascada, pero se siguen añadiendo funcionalidades según el cliente facilita nuevos requerimientos.

Esta metodología consiste en los siguientes pasos clásicos del desarrollo en cascada:

- **Análisis**: Primera fase del desarrollo, en este punto se extraen y estudian las necesidades y los requerimientos del usuario para poder poner en claro lo que se quiere.

- **Diseño**: Se estudia y descomponen las necesidades solicitadas por el usuario en la anterior etapa y se proyecta cómo se debe llevar a cabo el desarrollo de ese sistema. Que tecnologías y arquitecturas desarrollar para satisfacer el problema.

- **Implementación:** Una vez creado el sistema en esta fase se debe llevar a cabo la creación del sistema funcional, codificándolo como se ha sugerido en el anterior punto. En este punto se deben realizar determinadas pruebas, en concreto las unitarias, en otros puntos de este documento se describe en qué consisten estas pruebas.

- **Pruebas**: Una vez que está codificada la solución se debe validar que dicha implementación es correcta y cumple con todos los objetivos solicitados por el usuario.

- **Implantación**: La solución propuesta una vez probada se pone en producción, es decir se instala en las maquinas destinadas a tal efecto. Este paso no se ha afrontado dentro de este proyecto ya que al tratarse de un proyecto universitario pasara por un control previo de calidad antes de que sea implantado, pero se espera que pase a formar parte de un sistema mucho mayor que ya se encuentra en funcionamiento.

Entorno de trabajo

En este apartado se van a enunciar las herramientas que se han utilizado durante la realización del proyecto. Algunas de estas herramientas ya han sido descritas en el apartado "Estado del arte" ya que las propias herramientas que se han utilizado son la base del sistema.

Visual Studio 2008

Como ya se ha indicado en apartados anteriores .Net y más concretamente el lenguaje de desarrollo C# han sido las tecnologías elegidas para la codificación de la aplicación. Por ello Visual Studio ha sido el entorno de desarrollo integrado elegido para el desarrollo de la misma.

Dado que la versión final del sistema se ha desarrollado basándose en el framework 3.5 ha sido por lo tanto Visual Studio 2008 la versión utilizada. En el comienzo del proyecto se empezó codificando con el framework 2.0 y Visual Studio 2005 sin embargo a medida que avanzo el proyecto en el tiempo y apareció la nueva versión del Visual Studio se migro para poder utilizar la última versión existente.

Microsoft Office

Del paquete de ofimática de Microsoft se han utilizado.

Microsoft Office Word

Este procesador de texto ha sido utilizado para generar toda la documentación de este proyecto.

Microsoft Office Visio

Este software de dibujo se ha utilizado para crear los distintos diagramas UML incluidos en la memoria del proyecto.

Microsoft Office Proyect

Herramienta utilizada para la planificación de tiempos de proyectos, creada para ser utilizada en sistemas Windows.

Se ha utilizado dentro de este proyecto para la generación del diagrama de Gantt que describe los tiempos necesarios para la implementación del mismo, describiendo igualmente los distintos pasos y etapas por los que ha transcurrido el proyecto.

Windows 7 y Windows XP

Este sistema ha sido desarrollado bajo Windows XP al principio y bajo Windows Vista en la fase final de desarrollo. Aunque el sistema operativo no influye de manera importante en el desarrollo o ejecución del sistema, ya que estando implementado con .net es la versión del framework lo realmente importante, no hay que olvidar que la aplicación accede e interactúa con él ya que tiene que leer un fichero alojado localmente en la máquina host y escribir los ficheros descomprimidos en esa misma máquina, por lo que un cambió en las políticas de seguridad del S.O entre XP y Windows 7 podría haber influido.

Indexador de soluciones C# de Visual Studio .Net

En este apartado se va proceder a detallar las características fundamentales del indexador desarrollado. Es el punto más importante de toda la memoria ya que se explica en profundidad el análisis y diseño de la aplicación y por ende se explica en detalle el funcionamiento de la misma.

Primero se enuncia el análisis, pasando por los escenarios básicos y casos de uso y terminando con la especificación de requisitos.

A continuación se pasará a detallar el diseño centrándonos en las soluciones adoptadas y las tecnologías involucradas.

Análisis

Entorno Operacional

En este punto se describirá el entorno operacional en el que se deberá ejecutar la aplicación así como los requisitos que debe cumplir el mismo para que el funcionamiento del componente sea el deseado.

Tal y como ya se ha explicado anteriormente esta aplicación no es un componente independiente, sino que es parte de un sistema mucho mayor y más complejo que ha sido desarrollado por el departamento de informática de la universidad Carlos III y más concretamente por el grupo de "Knowledge Reuse". Así pues el entorno operacional de la aplicación está ligado en parte al entorno operacional del sistema matriz.

Aún así, y centrándonos únicamente en el indexador, existen una serie de requisitos que el entorno operacional debe cumplir para asegurar el correcto funcionamiento de la misma.

- **Ejecutarse sobre Windows**: El sistema operativo sobre el que se debe ejecutar la aplicación es Windows. Se han realizado pruebas con Windows XP, Windows Vista y Windows 7 siendo el funcionamiento totalmente satisfactorio.
- **Tener instalado Visual Studio 2008**: Es necesario tener instalada en la máquina la versión 2008 de Visual Studio para poder abrir la solución que se desea indexar y analizarla.
- **Tener instalado el framework 3.5:** Es necesario instalar el framework 3.5 para poder utilizar la biblioteca EnvDTE y EnvDTE80. No obstante con la instalación de Visual Studio 2008 se instala automáticamente este framework, por lo que aunque es un requisito indispensable se presupone que si se cumple el requisito anterior también se cumplirá este.

Estos son los tres requisitos que el entorno debe cumplir para poder ejecutar el componente. Destacar que no se ha enumerado como condición la instalación previa de ningún tipo de programa descompresor de archivos .zip ya que se utiliza una librería propia que ya va incluida en el proyecto. Así mismo para la utilización de nUML simplemente es necesario cargar una librería que el proyecto.

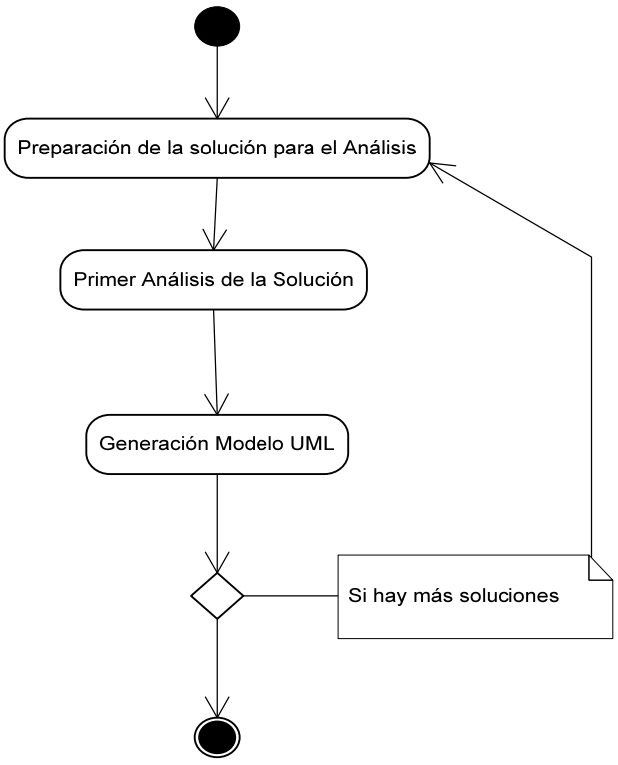
Especificación de Escenarios básicos y Casos de Uso

Dado que el programa desarrollado de indexación para soluciones C# pertenece a un sistema mucho más amplio y complejo y su existencia está ligado a su correcta integración con este, los casos de uso y escenario básicos no son demasiado numerosos quedando reducidos a las funcionalidad que la aplicación debe cumplir para el sistema matriz.

Aún así se han identificado los siguientes escenarios básicos y casos de uso de la aplicación que permiten abstraer la interacción del indexador con el usuario.

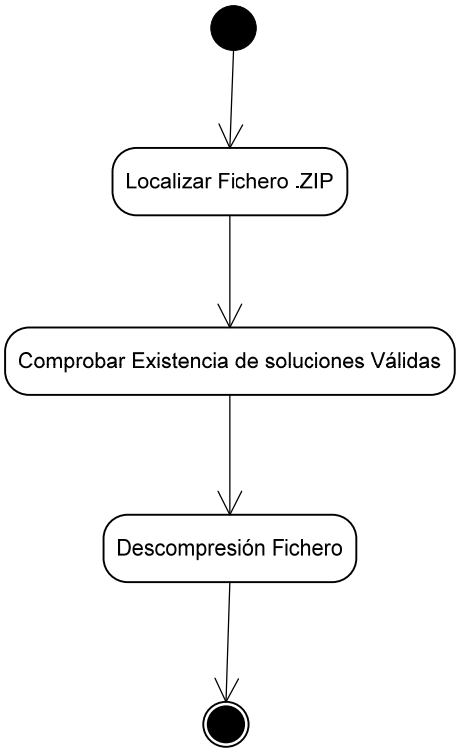
Escenarios básicos.

Este es el escenario principal de la aplicación.

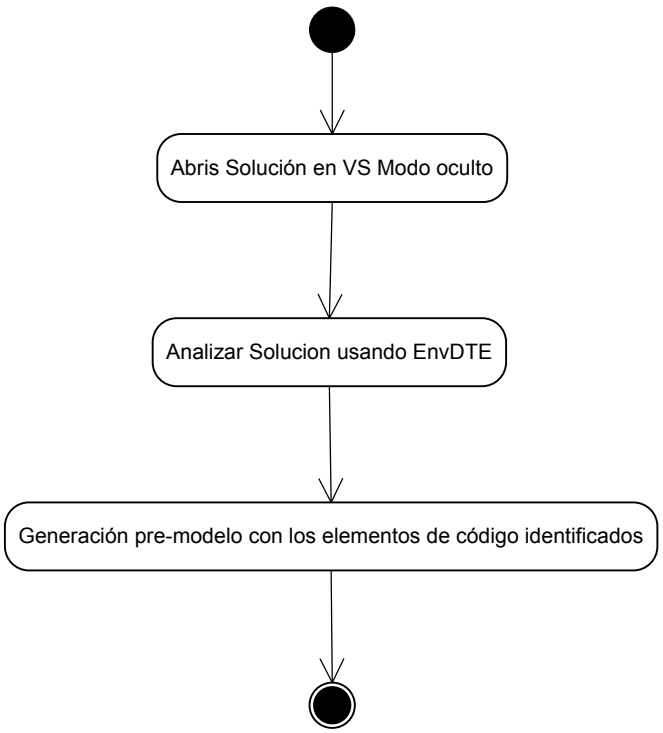


En el escenario básico de la aplicación que se describe en la figura 1 se pueden identificar tres sub-escenarios:

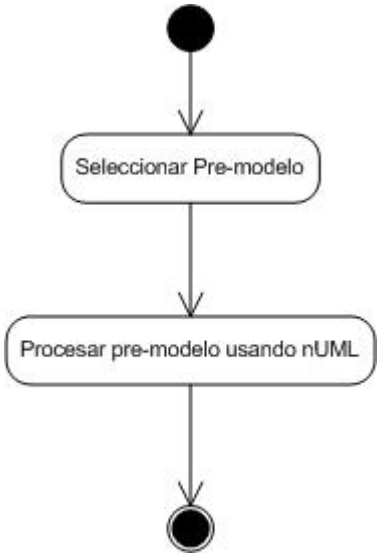
- ***Preparación de la solución para el análisis**:* Corresponde con el proceso de selección del fichero .zip que contendrá las soluciones a analizar, la comprobación de que existen soluciones válidas dentro de ese fichero y su posterior descompresión.



- ***Primer análisis de la solución:*** Una vez descomprimido el fichero .zip e identificadas las soluciones se procede a la fase de análisis de las mismas. En esta fase es cuando se identifican todos los elementos de código que aportan información relevante sobre la solución y que en la siguiente fase permitirán generar el modelo UML. Posteriormente se enumerarán más en detalle cuales son los elementos de código identificados por el indexador, ahora vamos a centrarnos en el proceso de obtención de esta información. Para ello se abre la solución utilizando la aplicación Visual Studio pero en forma oculta, de manera que aunque VS está siendo ejecutado la interfaz visual del mismo no está visible. Una vez abierta la solución utilizamos la biblioteca COM EnvDTE y EnvDTE80 para poder identificar los elementos de código presentes en la solución. Al final de esta fase obtendremos un pre-modelo con todos los elementos de código identificados.

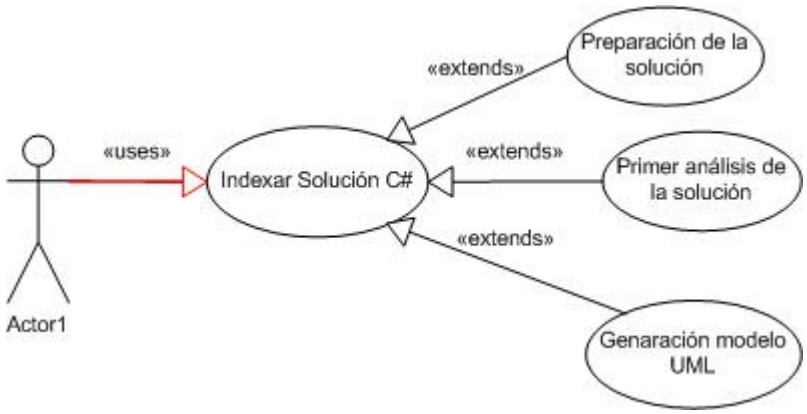


- ***Generación del modelo UML:*** Una vez que se han identificado los elementos de código que componen la solución se debe transformar estos datos en información. Par ello, y partiendo del pre-modelo generado en la fase anterior, usamos la biblioteca nUML para transformar este pre-modelo y recoger la información del modelo UML correspondiente en un fichero “xmi”.



Casos de Uso

A partir del escenario básico presentado en el apartado anterior se puede identificar el siguiente caso de uso.



Descripción textual de los casos de uso

En este apartado se describen textualmente el caso de uso de anterior. Con el fin de estructurar su análisis se han establecido, para cada uno de ellos, los siguientes atributos:

- **Identificador**: Se forma combinando las letras CU con un dígito secuencial
- **Actores**: actores que participan en el caso de uso
- **Objetivo**: Finalidad del caso de uso, es decir lo que se persigue con la ejecución de este caso de uso
- **Precondiciones**: Descripción del estado del sistema antes de la ejecución del caso de uso.
- **Postcondiciones**: Descripción del estado del sistema después de la ejecución del caso de uso.
- **Escenario básico**: Secuencia de acciones que se enlazan durante la ejecución del caso de uso.

CU-01	
Nombre	Indexación de soluciones.
Actores	Sistema matriz.
Objetivo	Analizar e identificar los elementos de código recogidos dentro de soluciones C# y generar un modelo UML
Precondiciones	• Debe existir un fichero .zip con la soluciones.
Postcondiciones	• Se generará un fichero “xmi” para recoger el modelo UML

	asociado a cada solución.
Escenario básico	1. Seleccionar fichero .zip 2. Analizar fichero .zip identificando la existencia de soluciones válidas 3. Descomprimir fichero .zip 4. Analizar e identificar los elementos de código de cada solución generando un pre-modelo. 5. Transformar los pre-modelos del paso anterior en un modelo UML.

Requisitos de usuario

A continuación se enumeran los requisitos de usuario asociados a la aplicación. Cada requisito representa "una condición o una funcionalidad exigidas por el usuario para resolver un problema o para conseguir un objetivo", por lo que según esta definición, es posible clasificar a su vez los requisitos en dos categorías: requisitos de capacidad y requisitos de restricción.

Para cada uno se han fijado además los siguientes atributos:

- **Identificador de requisito**: cada requisito debe ser identificable unívocamente. Para los requisitos de capacidad se utiliza el formato RUC-XX y para los de restricción RUR-XX (siendo XX un número natural de dos cifras).

- **Necesidad**: los requisitos pueden clasificarse en esenciales, opcionales y convenientes según su importancia.

- **Estabilidad**: un requisito que no es estable presenta una gran dependencia de las fases posteriores del proceso de desarrollo y tiene una alta probabilidad de ser modificado en el futuro según se vaya profundizando en dichas fases.

- **Prioridad**: este atributo establece en qué momento respecto al resto de requisitos debería estar disponible la parte del sistema a la que se refiere el requisito.

- **Fuente**: informa acerca del origen del requisito.

Requisitos de capacidad

Los siguientes requisitos enumeran la funcionalidad del sistema.

RUC-01			
Descripción	La aplicación recibirá como fichero de entrada un fichero comprimido con extensión .zip que contendrá las soluciones a analizar.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

RUC-02			
Descripción	La aplicación utilizará Visual Studio para leer las soluciones escritas en lenguaje C# encontradas en el fichero .zip.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

RUC-03			
Descripción	Por cada solución se debe extraer la máxima información posible.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

RUC-04			
Descripción	El objetivo final de la aplicación es generar un modelo UML con la información indexada.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

RUC-05			
Descripción	Si en el fichero .zip de entrada existen varias soluciones se deberán analizar todas y cada una de ellas de forma independiente.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

RUC-06			
Descripción	Por cada solución encontrada se generará un modelo UML.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

Requisitos de restricción

Los siguientes requisitos afectan a la forma en la que el sistema realiza sus operaciones.

IDENTIFICADOR RUR-01			
Descripción	La dll que resulte tiene que implementar la interfaz nUML. Transform proporcionado por el departamento.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

IDENTIFICADOR RUR-02			
Descripción	El tipo de bmodelo que se debe devolver es de tipo UML.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

IDENTIFICADOR RUR-03			
Descripción	La dll resultado debe estar implementada en la plataforma .net		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

IDENTIFICADOR RUR-04			
Descripción	La dll resultado ejecutarse en entorno Windows.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente	Estabilidad	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	Cliente		

Requisitos software

Una vez identificados todos los requisitos de usuario de la aplicación es posible enumerar los requisitos software que están reflejados en ellos. Los requisitos software nos dan una definición exacta del sistema y son el punto de partida para las posteriores fases de análisis y diseño.

Para su descripción se ha utilizado la misma plantilla de atributos que en el caso de los requisitos de usuario, pero además, se ha añadido información acerca de la **verificabilidad** de cada requisito, que mide en qué grado resulta posible comprobar si se está cumpliendo un requisito una vez implementado en el sistema.

El formato de los identificadores en este caso es diferente también: los requisitos de software funcionales utilizan **RSF-XX**, mientras que los no funcionales usan **RSN-XX**.

Requisitos funcionales

IDENTIFICADOR: RSF-01			
Descripción	Se recibirá un path que contenga la localización del fichero .zip.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-01		

IDENTIFICADOR: RSF-02			
Descripción	Es sistema debe realizar la descompresión del fichero .zip.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-01		

IDENTIFICADOR: RSF-03			
Descripción	Antes de realizar la descompresión se debe comprobar que existen soluciones válidas. Si no es así no se realiza la descompresión y la ejecución del la aplicación se da por finalizada.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-01		

IDENTIFICADOR: RSF-04			
Descripción	Si existen varias soluciones cada una se analizara de forma independiente.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-05		

IDENTIFICADOR: RSF-05			
Descripción	Cada solución encontrada se abrirá en la instancia de Visual Studio 2008 creada.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-02, RUC-05		

IDENTIFICADOR: RSF-06			
Descripción	No existirán simultáneamente varias instancias de Visual Studio 2008 ejecutándose al mismo tiempo. Solo se creará una instancia de Visual Studio 2008 sobre la que se irán cargando las soluciones.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-02, RUC-05		

IDENTIFICADOR: RSF-07			
Descripción	Se deben *matar* todas las instancias de Visual Studio 2008 que se hayan abierto para analizar una solución. Teniendo especial cuidado en el caso de que se haya producido una excepción durante la ejecución.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-02		

IDENTIFICADOR: RSF-08			
Descripción	Visual Studio 2008 se ejecutará de forma oculta de manera que aunque el programa se está ejecutando y tenga cargada una solución el usuario no verá la interfaz de Visual Studio 2008.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-02		

IDENTIFICADOR: RSF-09			
Descripción	Por cada solución se deben obtener todas las clases que existan		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-10			
Descripción	Por cada solución se deben obtener todos los interfaces.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-11			
Descripción	De cada clase se debe obtener: • Nombre • Visibilidad • Si es abstracta o no • Comentarios		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-12			
Descripción	Por cada clase se tiene que obtener todos sus atributos.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-13			
Descripción	Por cada clase se tiene que obtener todos sus métodos.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-14			
Descripción	Por cada clase se tiene que obtener todas las dependencias que existan.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-15			
Descripción	Por cada clase se tiene que obtener todas las asociaciones que existan.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-16			
Descripción	Por cada clase se tiene que obtener todas las generalizaciones que existan.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-17			
Descripción	De cada interfaz se debe obtener: • Nombre • Visibilidad • Comentarios existan		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-18			
Descripción	Por cada interfaz se deben obtener todos sus métodos.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-19			
Descripción	Por cada interfaz se debe obtener todas sus dependencias.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-20			
Descripción	En cada atributo se debe identificar: • Nombre • Tipo • Visibilidad • Comentarios		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-21			
Descripción	En cada método se debe identificar: • Nombre del método • Tipo de método (si es constructor o no) • Si está sobrecargado • Visibilidad • Comentarios • Valor de retorno • Parámetros		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-22			
Descripción	En cada parámetro se debe identificar: • Nombre • Tipo		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-03		

IDENTIFICADOR: RSF-23			
Descripción	Con toda la información especificada en los requisitos: RSF09, RSF10, RSF11, RSF12, RSF13, RSF14, RSF15, RSF16, RSF17, RSF18, RSF19, RSF20, RSF21 y RSF22 se debe generar un modelo UML que la recoja.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-04		

IDENTIFICADOR: RSF-24			
Descripción	Se generarán tantos modelos UML de salida como soluciones se hayan analizado		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUC-06		

Requisitos no funcionales

IDENTIFICADOR: RNF-01			
Descripción	El código de la aplicación debe implementar la interfaz nUML. Transform.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUR-01		

IDENTIFICADOR: RNF-02			
Descripción	Se generará un fichero “xmi” que permita recoger el modelo UML.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUR-02		

IDENTIFICADOR: RNF-03			
Descripción	Se utilizará .NET como plataforma de desarrollo.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUR-03		

IDENTIFICADOR: RNF-04			
Descripción	Se utilizará Visual Studio 2008 como IDE de desarrollo.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUR-03		

IDENTIFICADOR: RNF-05			
Descripción	Windows será el entorno de ejecución de la aplicación.		
Necesidad	☑Esencial ☐Opcional ☐Conveniente		
Estabilidad	☑ Sí ☐ No	Verificable	☑ Sí ☐ No
Prioridad	☐ Alta ☑ Media ☐ Baja		
Fuente	RUR-04		

Trazabilidad de requisitos

A continuación se muestra la tabla de trazabilidad de requisitos para el análisis del sistema.

	RUC-01	RUC-02	RUC-03	RUC-04	RUC-05	RUC-06	RUR-01	RUR-02	RUR-03	RUR-04
RSF-01	●									
RSF-02	●									
RSF-03	●									
RSF-04					●					
RSF-05		●			●					
RSF-06		●			●					
RSF-07		●								
RSF-08		●								
RSF-09			●							
RSF-10			●							
RSF-11			●							
RSF-12			●							
RSF-13			●							
RSF-14			●							
RSF-15			●							
RSF-16			●							
RSF-17			●							
RSF-18			●							
RSF-19			●							
RSF-20			●							
RSF-21			●							
RSF-22			●							
RSF-23				●						
RSF-24						●				
RSN-01							●			
RSN-02								●		
RSN-03									●	
RSN-04									●	
RSN-05										●

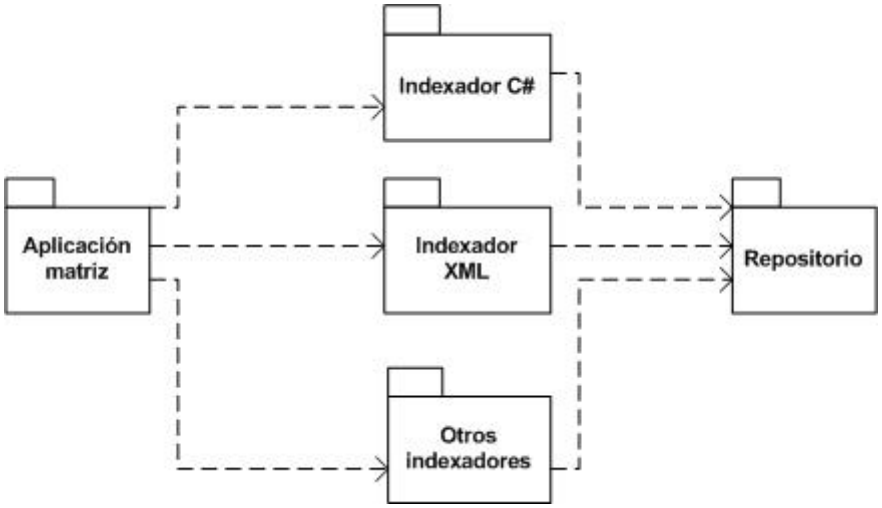
Diseño

Hasta el momento se ha realizado el análisis de la aplicación, se ha comenzado introduciendo los casos de uso y escenarios básicos del sistema como punto de partida para realizar el análisis. A continuación se han enumerado los requisitos de usuario, tanto de capacidad como de restricción, que el sistema debe cumplir y se ha terminado identificando los requisitos software. Resumiendo hemos establecido “Que debe hacer el sistema”.

Durante este apartado definiremos “Como funciona el sistema”, es decir la estructura interna que permite a la aplicación dar la funcionalidad deseada. Primero se mostrará el diseño arquitectónico del indexador y a continuación se describirá en detalle la estructura interna del sistema para terminar con el diagrama de clases que representa a la aplicación.

Diseño arquitectónico

A continuación se muestra cómo se integra el componente que se proporcionará, dentro de la arquitectura global de la aplicación.



Como se observa en el diagrama, el presente proyecto se basa en la creación de un indexador de C#. Este componente no funciona por si solo si no que debe encuadrarse dentro del marco de trabajo de una aplicación superior. Para una mejor comprensión se describirá cada uno de los componentes del diagrama:

- **Aplicación superior:** Esta aplicación controla la gestión de los indexadores encargados de ir obteniendo la información de las distintas fuentes. Estas fuentes pueden ser muy variadas: ficheros XML, HTML o de código, en resumen todo aquello que aporte información semántica aprovechable.
- **Indexador de C#:** El indexador objeto de este proyecto.
- **Indexador XML:** Ejemplo de Indexador existente dentro del marco de trabajo. En este caso el origen de la información es diferente pero el resultado debe ser el mismo, un modelo UML representado a través de un fichero “xmi”.
- **Otros Indexadores.** Debido al diseño de la aplicación superior, este permite la integración de nuevos Indexadores de manera que se pueda indexar la mayor información posible que se puede encontrar dentro del heterogéneo mundo de aplicaciones informáticas.

- **Repositorios:** Es también parte de la aplicación matriz en la que se encuentra englobada el sistema. Permite almacenar la base de conocimiento generada por los indexadores.

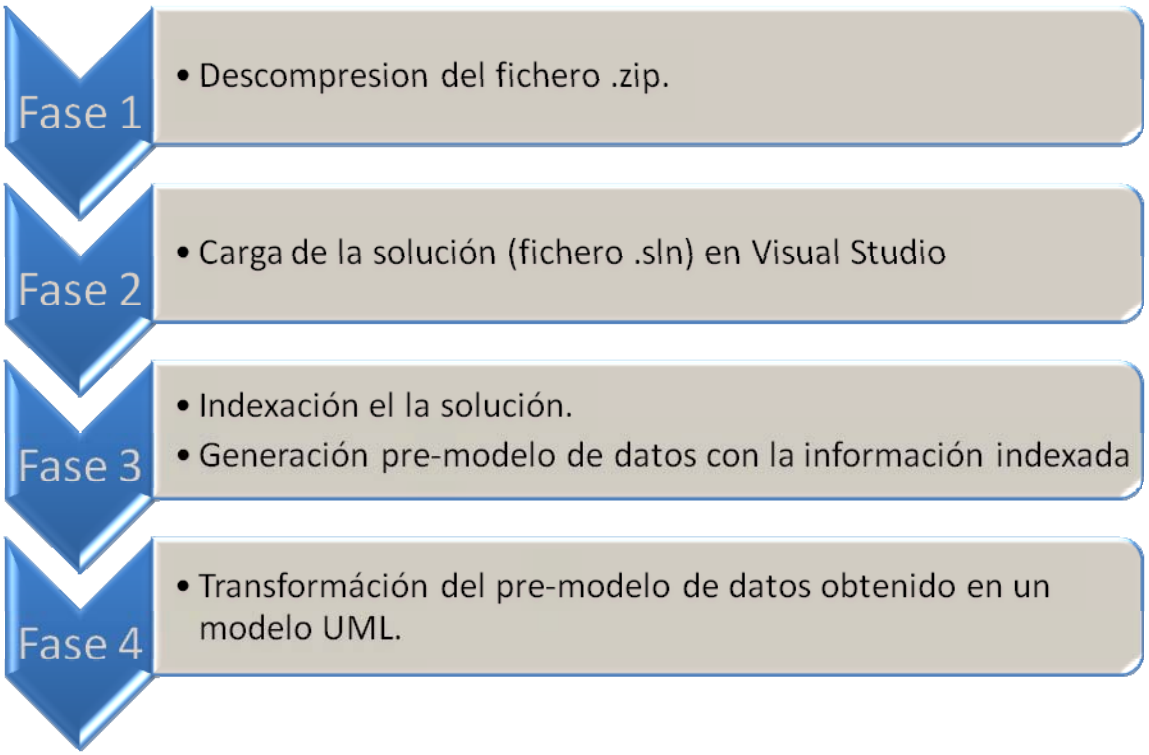
Arquitectura interna

En este apartado se describirá con detalle la arquitectura interna del sistema, es decir cómo funciona, cuales son las fases de ejecución y qué relación existe entre ellas, así como cuales son las tecnologías involucradas en cada fase y que han permitido realizar el desarrollo del sistema.

Con el objetivo de poder indexar la información presente en una solución de Visual Studio, teniendo además la dificultad añadida de que las soluciones fuentes vienen comprimidas en un fichero “zip”, se debe trabajar con diferentes tecnologías que de forma combinada permitan alcanzar los objetivos planteados.

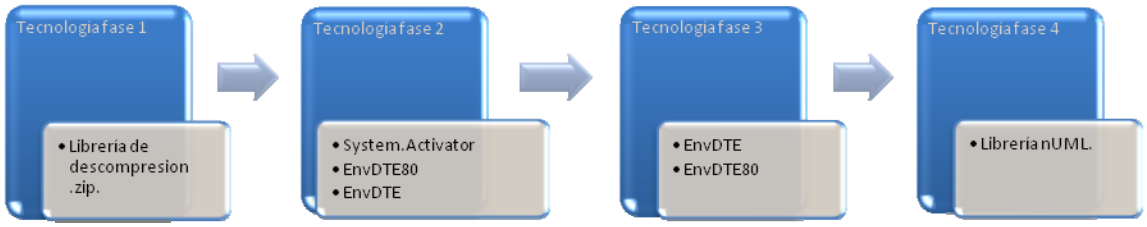
De manera muy general se pueden identificar cuatro grandes bloques de diseño dentro de la aplicación, que corresponden con las cuatro fases principales de ejecución de la misma.

5. Fase de descompresión del fichero .zip.
6. Fase de carga de la solución (o soluciones) en Visual Studio.
7. Fase de indexación de de la información contenida en la solución.
8. Fase de generación del modelo UML, representado a través de un fichero “xmi”, utilizando la información obtenida en la fase anterior.


Fase 1
• Descompresion del fichero .zip.
Fase 2
• Carga de la solución (fichero .sln) en Visual Studio
Fase 3
• Indexación el la solución.
• Generación pre-modelo de datos con la información indexada
Fase 4
• Transformáción del pre-modelo de datos obtenido en un modelo UML.

En la anterior figura se pueden identificar las cuatro fases principales de ejecución de la aplicación que van asociadas con los cuatro principales bloques de diseño que se ha tenido que abordar durante la realización del sistema. Si bien toda la aplicación ha sido desarrollada bajo .NET para cada uno de los bloque de diseño ha sido necesario el uso de una o varias bibliotecas que han jugado un papel clave y que han permitido poder alcanzar los objetivos marcados.

En el siguiente gráfico se muestran asociadas a cada fase las tecnologías clave aplicadas.



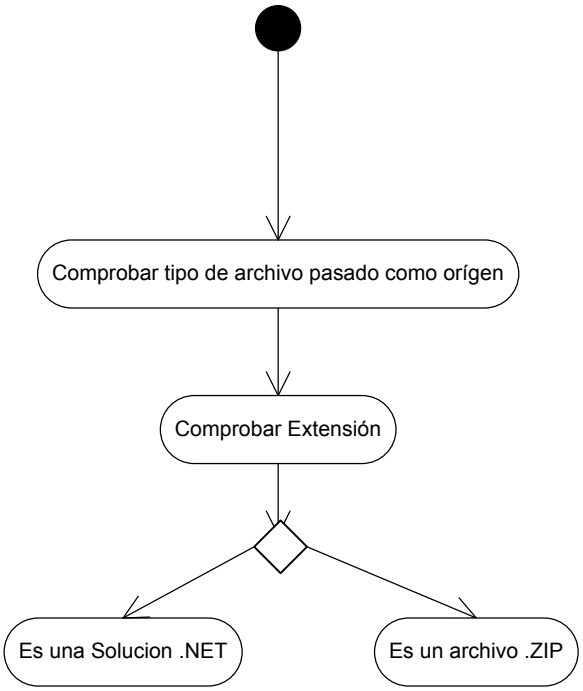
A continuación se explicará con más detalle cada una de estas fases y las bibliotecas usadas. Se analizará cual ha sido la solución de diseño adoptada y la relación entre todas ellas.

Descompresión fichero .zip

Durante este apartado se explicará el proceso de descompresión que el indexador debe realizar para poder obtener las soluciones. Se comentarán las tecnologías utilizadas y su aplicación en el sistema.

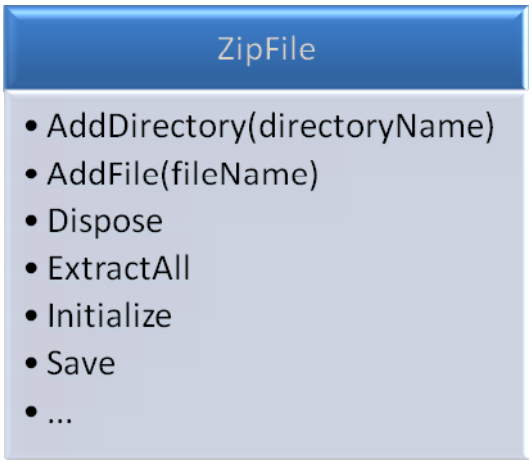
Como requisito indispensable el sistema debe ser capaz de descomprimir los ficheros .zip que se le pasen como entrada para poder acceder a las soluciones que contenga.

Como se comenta a lo largo de esta documentación el componente debe aceptar el paso de archivos .ZIP y de soluciones .NET. Lo primero que hace el indexador será el validar mediante la extensión del archivo pasado a qué grupo pertenece. Este punto solo aplicará para aquellos archivos pasados que tengan una extensión .ZIP.

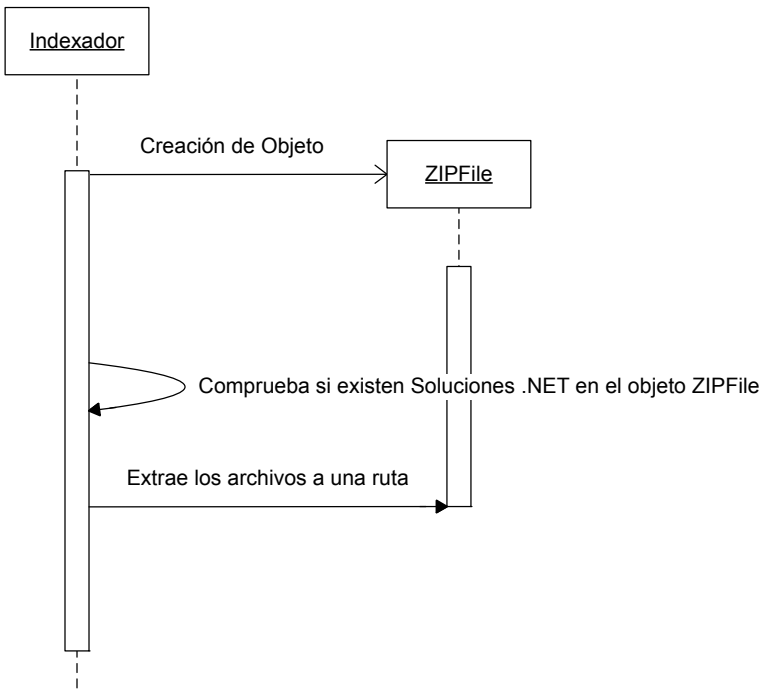


Se desea depender lo menos posible de aplicaciones externas por lo que es recomendable no poner como requisito indispensable del entorno la instalación de una aplicación de descompresión de archivos. Por y para ello se buscó a través de la red algún tipo de librería que facilitase ese trabajo y que simplemente referenciándola realizase la tarea solicitada.

El componente encontrado llamado Ionic.ZIP, es una librería libre que proporciona las rutinas necesarias para la descompresión de archivos .ZIP.



A continuación se muestra la forma en la que el indexador descomprime los archivos:



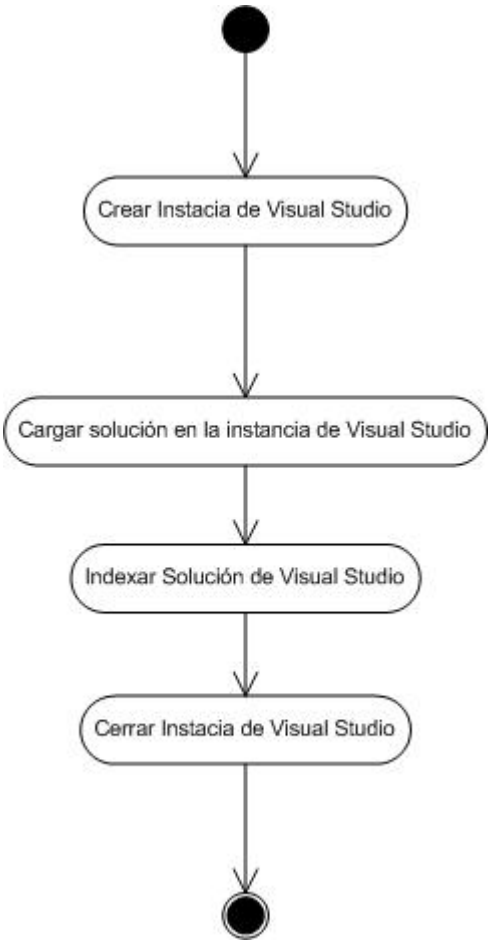
Este diagrama esta realizado para el caso en que el indexador encuentre soluciones .ZIP dentro del fichero comprimido en caso contrario, es decir el archivo .ZIP que se pasa no contiene soluciones .NET, el Indexador no extrae los ficheros del archivo .ZIP y acaba su ejecución.

Creación de la instancia de Visual Studio y carga de la solución

En este apartado se explicará en detalle el proceso de creación de la instancia de Visual Studio y como se carga en esta instancia las soluciones encontradas en el fichero .zip.

Una vez descomprimido el fichero .zip es posible comenzar a manipular los ficheros .sln asociados a la solución. Para poder acceder a toda la información que la solución contiene es necesario crear una instancia de Visual Studio y abrir en ella las soluciones encontradas. Una vez abierta las soluciones en la instancia de Visual Studio es posible empezar la indexación. Hay que aclarar que en el caso de que el fichero .zip contuviera más de una solución no se abren simultáneamente todas en la instancia de Visual Studio si no que primero se carga una y tras finalizar el proceso de análisis se carga las siguiente.

En el siguiente gráfico se pueden ver las secuencias de acciones en las que está involucrada la instancia de Visual Studio en el marco de ejecución del sistema



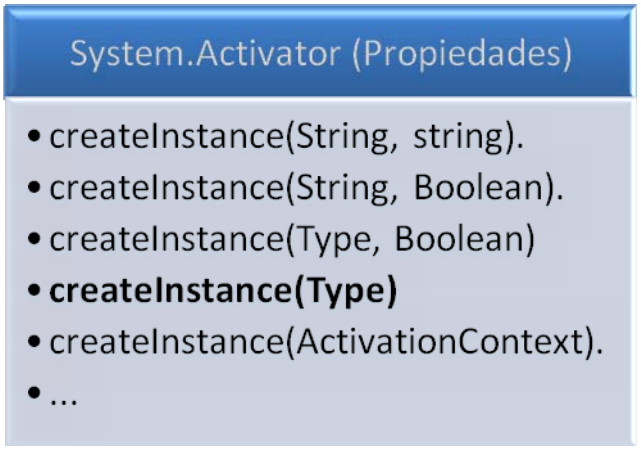
Por lo tanto en esta fase se pueden diferenciar tres procesos fundamentales relacionados con Visual Studio: creación de la Instancia de Visual Studio, carga de la solución de la instancia de Visual Studio y cierre de Visual Studio.

1. **Creación de la instancia de Visual Studio:** Primero es necesario lanzar en background y de forma totalmente trasparente al usuario una instancia de Visual Studio, de manera que desde el sistema tengamos acceso a esta instancia y sea posible manipularla con total libertad.
2. **Carga de la solución:** Una vez creada la instancia de Visual Studio tiene que ser posible cargar las soluciones a analizar para poder tener acceso a todos los elementos que la componen y en especial a los ficheros de código que son el objetivo del análisis.
3. **Cierre de Visual Studio:** Si no existen más soluciones a analizar se cierra la instancia de Visual Studio, en caso contrario se vuelve al punto 2.

En los siguientes apartados se explicará con más detalle estas fases, comentando el funcionamiento del sistema en cada uno de estos procesos, así como los problemas y las soluciones tecnológicas adoptadas. La fase “Indexar Solución de Visual Studio” que aparece en el gráfico anterior, será explicada con más detalle más adelante.

Creación instancia Visual Studio

Para poder crea una instancia de Visual Studio en tiempo de ejecución se ha utilizado la clase **Sytem.Activator**. Esta clase permite crear tipos de objetos localmente y obtener una referencia al mismo para poder manipularlo.



De entre todos los métodos que la clase ofrece se ha utilizado *createInstance(Type*), donde *type* indica el tipo de objeto que queremos crear, que en este caso es "Visual.Studio.DTE.9.0". El método devolverá una referencia al objeto creado para que sea posible manipularlo. Dado que el tipo de objeto que se creara es"Visual.Studio.DTE.9.0" se obtendrá una referencia a un objeto de tipo *EnvDTE80.DTE2* que es el objeto de nivel superior del modelo de objetos de automatización de Visual Studio.

De esta manera se ha creado una instancia de Visual Studio sobre la que abrir la solución y además es posible acceder a ella para configurarla según el criterio deseado. Por último solo es necesario modificar dos propiedades del objeto DTE2 para asegurar que la instancia que se ha creado permanezca en modo oculto y su ejecución sea totalmente trasparente.

Estas propiedades son:

- **SuppressUI**: Esta propiedad obtiene o establece un valor que indica si se debe mostrar la interfaz de usuario durante la ejecución de código de automatización de Visual Studio. En este caso asignaremos el valor true para que la interfaz no se muestre.
- **UserControl**: Obtiene un valor que indica si el entorno fue iniciado por un usuario o por automatización. En este caso al ser iniciado por automatización asignamos el valor false.

Gracias a esto tendremos una instancia de Visual Studio 2008 ejecutándose en la máquina host pero de manera totalmente transparente al usuario.

Para asegurar que no quedan procesos de Visual Studio ejecutándose en la máquina host una vez que ya no sean necesarios, utilizaremos el método *"Quit*" para terminar con la ejecución y liberar todos los recursos usados.

Apertura de la solución en la instancia de Visual Studio creada.

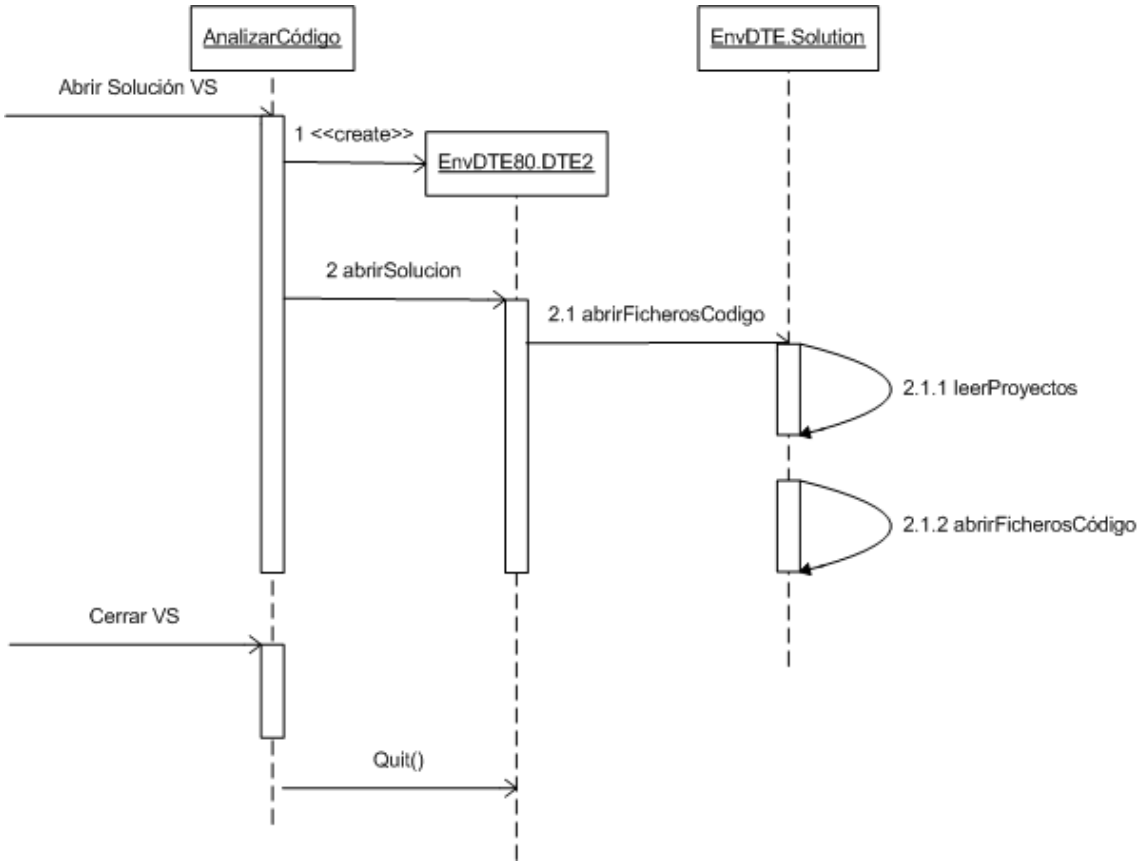
El siguiente paso antes de comenzar la indexación es abrir la solución en la instancia de Visual Studio creada. La clase *EnvDTE.Solution* permite crear un objeto de tipo solución asociado al objeto de Visual Studio, de esta forma una vez creado un objeto de tipo *Solution* se puede utilizar el método "*Open*" , pasándole como parámetro la ruta del fichero .sln para abrir finalmente la solución.

Es necesario realizar un último paso para poder tener acceso total a la solución y a todos sus recursos. Hasta ahora se ha conseguido cargar la solución y todos los proyectos que contenga en la instancia de Visual Studio, sin embargo todavía no se tiene acceso a los ficheros de código contenidos dentro de los proyectos. Sin este acceso no podemos realizar la indexación ya que son los ficheros con extensión "cs" los que contienen toda la información a la que queremos acceder. Para poder analizar el código de estos ficheros es necesario abrirlos de forma explícita esto es, se debe acceder uno por uno a todos los elementos del proyecto y en el caso de que sea un fichero de código utilizar el comando "Open" de la clase *EnvDTE.ProjectItem* para cargarlo.

A partir de aquí, una vez que todos los ficheros de código encontrados en la solución han sido abiertos, se puede continuar con la siguiente la fase de indexación.

Diagrama de secuencia de la carga de la solución en Visual Studio

El diagrama de secuencia para esta fase sería el siguiente.

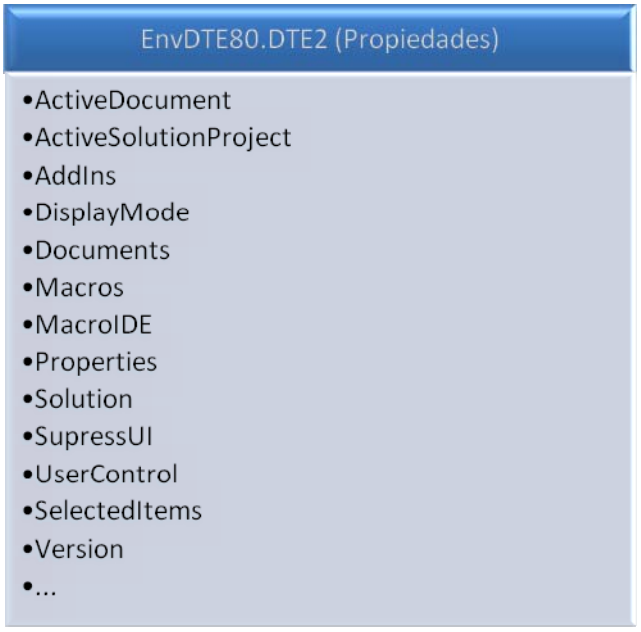


Acceso a los ficheros de código.

En esta apartado se explicará el proceso seguido para poder acceder al contenido de los ficheros de código abiertos, es decir, como a través de la instancia de Visual Studio obtenemos el control, no solo de los ficheros de código sino de su propio contenido.

Una vez abierta la solución en Visual Studio, y habiéndose asegurado de que todos los ficheros de código han sido cargados correctamente se puede comenzar a obtener la información que contiene. Para ello es necesario que, de entre todos los ficheros que han sido cargados con la solución se seleccionen para indexar solo aquellos que aportan valor semántico, es decir los ficheros de código.

El método *AnalizarCódigo*, será el encargado de comenzar el proceso de recopilación de información recibiendo como parámetro la referencia a la instancia de Visual Studio creada. Como se ha indicado anteriormente la instancia de Visual Studio se representa gracias a la interfaz DTE2.



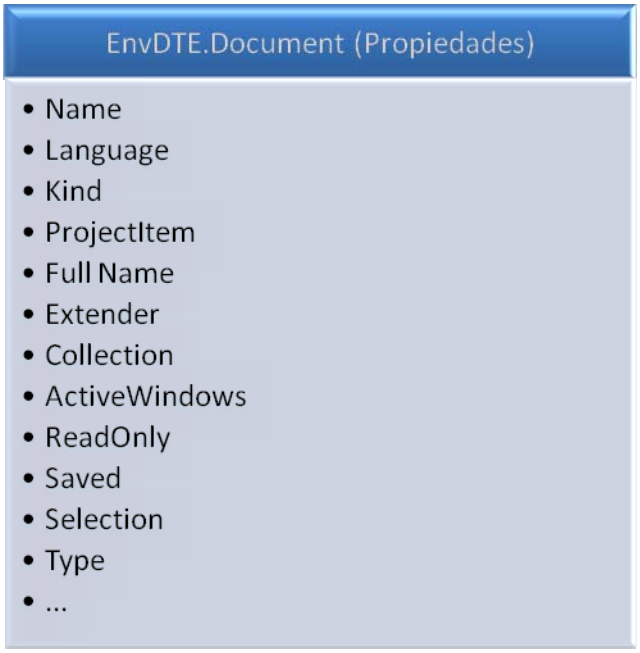
De entre todas las propiedades a las que tenemos acceso desde la interfaz DTE2 cabe destacar las siguientes:

- **ActiveDocument**: Obtiene el documento activo.
- **ActiveSolutinProjects**: Obtiene una matriz de proyectos seleccionados actualmente.

- **AddIns**: Obtiene la colección de addIns, que contiene todos los complementos disponibles.
- **Documents**: Obtiene la colección de documentos abiertos en el entorno de desarrollo
- **Solution**: Obtiene el objeto Solution que representa todos los proyectos abiertos en la instancia actual del entorno y permite el acceso a los objetos de generación.
- **SupressUI**: Obtiene un valor que indica si se debe mostrar la interfaz de usuario.
- **UserControl**: Obtiene un valor que indica si el entorno fue iniciado por un usuario o por automatización.
- **Version**: Obtiene el número de versión de la aplicación host.

Gracias a la propiedad *Documents* es posible obtener la colección de todos los documentos abiertos en la instancia de Visual Studio, de esta forma hemos pasado de manejar la instancia de Visual Studio a una colección de Documentos abiertos en la solución.

De la clase *Document* se puede comenzar a extraer información útil y que aporta valor semántico. La siguiente figura muestra las propiedades más importantes de esta clase.

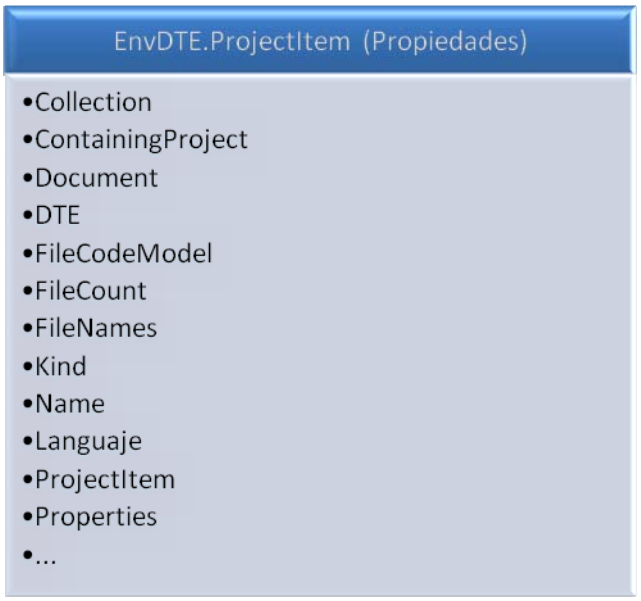


De entre estas propiedades destacan:

- **Name**: Obtiene el nombre del documento.
- **Kind**: Obtiene el tipo de documento.
- **Language**: Obtiene el lenguaje del documento. En este caso permitirá saber si el documento está escrito en C# o VB por ejemplo.

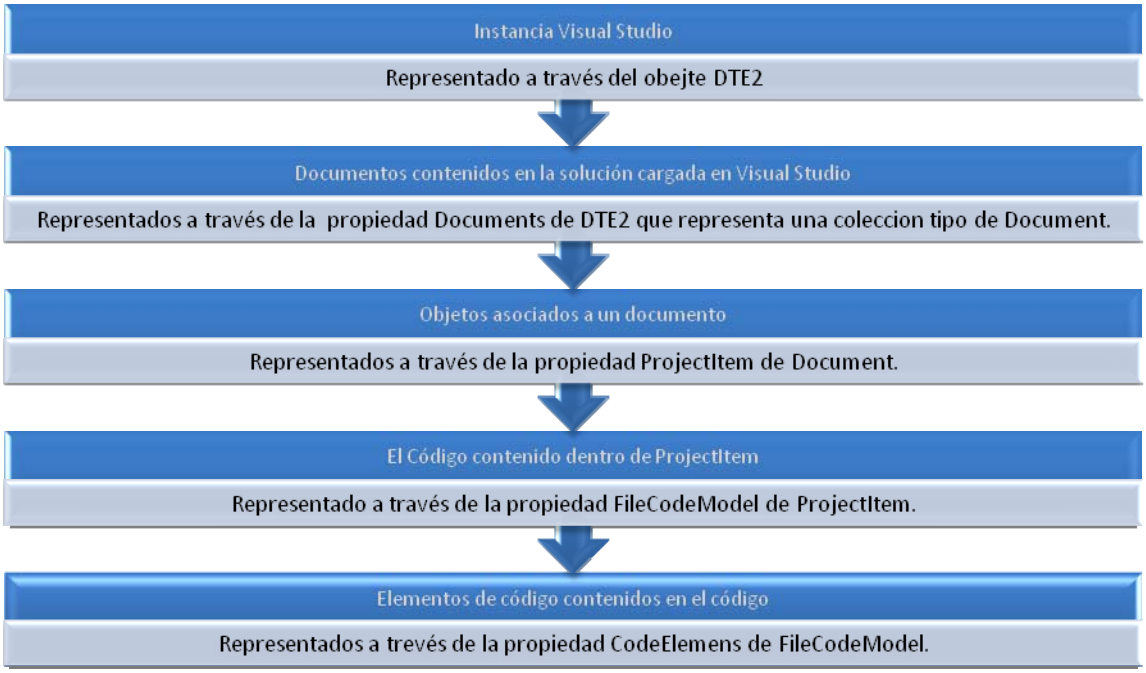
- **ProjectItem**: Obtiene los objetos de proyectos asociados al documento.

Todavía es necesario profundizar más para poder tener acceso a toda la información contenida en el fichero de código y esto se consigue accediendo a la propiedad *ProjectItem* que devuelve todos los objetos asociados al documento.



De entre todas las propiedades de esta clase solo es relevante una: *FileCodeModel*. Esta clase representa el código contenido dentro de un documento de código y es por lo tanto esta clase, la que contiene toda la información que la aplicación debe indexar. Toda esta información se encuentra recogida dentro de la clase *FileCodeModel* en su propiedad *CodeElements* que devuelve una colección con todos los elementos de código.

La siguiente figura muestra el proceso seguido desde la instancia de Visual Studio hasta poder acceder a los elementos de código.



De esta forma a partir de los *CodeElements* podemos comenzar el proceso de indexación del documento. Este proceso consistirá en la identificación de los *CodeElements* que representan la información que la aplicación debe ser capaz de reconocer y que ha sido detallada en la especificación de requisitos.

Indexación de la información de los ficheros de código

Llegamos a una de las fases de ejecución clave del sistema, ya que es ahora cuando se analiza e identifican en el código todos aquellos elementos que aportan información semántica válida y que por lo tanto serán utilizados para generar el modelo UML. Nada que no sea identificado en esta fase será transformado usando nUML y por lo tanto una análisis ineficiente provocará que se genere un fichero "xmi" que no describa correctamente el modelo UML asociado a la solución.

Tras la identificación de estos elementos se generará un pre-modelo intermedio que permitirá almacenar toda la información indexada y que será el punto de entrada al proceso de conversión hacia el modelo UML.

En este apartado se explicara el proceso seguido para identificar todos aquellos elementos de código y relaciones listados en el apartado de requisitos.

Una vez llegado hasta los *CodeElement*, los mínimos elementos de información que somos capaces de analizar, es posible comenzar con el proceso de indexación. El hecho de que se esté trabajando ya con elementos de un nivel tan bajo de abstracción permite por fin identificar en los ficheros de código aquellos elementos de información que fueron especificados en los requisitos.

Este es un paso crítico para el funcionamiento del sistema ya que del correcto análisis e identificación de elementos y relaciones presentes en el código dependerá la generación del modelo UML. Si no se consigue generar un pre-modelo de información completo el fichero "xmi" generado no recogerá toda la información semántica que los ficheros de código ofrecen.

Esencialmente este proceso de análisis en el que se fundamente la fase de indexación está basado en:

- *Identificación de los elementos de código*: Elementos que componen por sí solos el código y que son identificables en primera instancia, tales como: clases, atributo o métodos.
- *Identificación de relaciones existentes en el código*: Relaciones que se dan entre los elementos de código identificados tales como: dependencias, generalizaciones o asociaciones.

Identificación de los elementos de código

Los elementos de código y las características principales de los mismos que el indexador es capaz de reconocer son:

- **Clases**
  - *Atributos*: Dentro de los atributos ser recoge el nombre, tipo y visibilidad.
  - *Nombre*: Nombre de la clase.
  - *Visibilidad*: Visibilidad de la clase si han declarado alguna. Si no se declara visibilidad EnvDTE entiende que la visibilidad es privada. Esta no es una decisión nuestra si no de la propia librería de Microsoft.
  - Si es abstracta o no
  - *Métodos*: Dentro de cada método se recoge el valor de retorno, nombre, visibilidad, parámetros, tipo (si es una función constructora o no), si está sobrecargado.

- **Interfaces**
  - *Nombre*: Nombre de la interface.
  - *Visibilidad*: Visibilidad de la interface.
  - *Métodos*: Los métodos que están definidos dentro de la interface. Las características asociadas a los métodos son las mismas que en el caso de las clases.

Estos elementos de código y sus características son directamente identificable y están relacionados uno a uno con cada *CodeElement*, es decir cada *CodeElement* representa un elemento de código. De esta forma el proceso de identificación consiste en comprobar que la clase de un *CodeElement* se corresponde con alguno de los elementos de información que el indexador tiene que ser capaz de identificar.

Para poder realizar este filtrado entre los *CodeElement* y recoger únicamente la información de aquellos que son útiles usamos la propiedad *Kind*. La siguiente tabla muestra la relación entre la propiedad *Kind* y los elementos de información que el sistema debe identificar.

KIND	UNIDADES DE INFORMACIÓN
vsCMElement.vsCMElementNamespace	Namespace
vsCMElement.vsCMElementVariable	Atributo de Clase
vsCMElement.vsCMElementClass	Clase
vsCMElement.vsCMElementInterface	Interfaz
vsCMElement.vsCMElementFunction	Metodo

Estos son los cinco tipos principales de unidades de información que identificamos en los CodeElement. Profundizando en ellos se puede obtener más información que en muchos casos el indexador tiene que ser también capaz de identificar.

La siguiente tabla muestra de manera más ampliada la unidad primaria de información asociada con los elementos de información que se pueden obtener de ellos.

KIND	UNIDADES INFORMACION	ELEMENTOS INFORMACION ASOCIADOS
vsCMElement.vsCMElementVariable	Atributo	Nombre Tipo Visibilidad Si es tipo primitivo Comentarios
vsCMElement.vsCMElementClass	Clase	Nombre Si es Abstracta Visibilidad Comentarios
vsCMElement.vsCMElementInterface	Interfaz	Nombre Visibilidad Comentairos
vsCMElement.vsCMElementFunction	Metodo	Nombre Visibilidad Parametro salida Parametros entrada

Identificación de las relaciones existentes en el código

Una vez identificados los elementos de código por los que están compuestos los ficheros .cs de la solución se deben establecer las relaciones existentes entre ellos, pues aportan también un valor semántico muy importante al resultado del análisis.

El indexador de código C# debe de ser capaz de reconocer las siguientes relaciones:

- **Clase**
  - *Generalizaciones*: En el caso de que esté heredando de una o más clases.
  - *Asociaciones*: En el caso de que la clase tenga alguna asociación con alguna otra clase.
  - *Agregaciones*: En el caso de que la clase tenga una relación de agregación con alguna otra clase.
  - *Implementaciones*: En el caso de que la clase este realizando alguna implementación de una interface.
  - *Dependencias*: En el caso de que existan dependencias con otras clases.

En este caso la identificación de las relaciones no es directa como en el caso de los elementos de código, sino que viene dada por la aplicación del conocimiento a la hora de codificar el sistema. De esta forma de cada elemento de código y sus características se pueden inferir las siguientes relaciones.

- **Atributo**: Permite identificar asociaciones y agregaciones en el caso de que su tipo no sea primitivo.
- **Clase**: De las características de la clase se pude determinar si existe generalización o implementa interfaces
- **Método**: De los métodos incluidos dentro de las clases se pueden identificar las dependencias.

Generación del modelo UML representado mediante un fichero "xmi"

En este apartado se explicara cómo se usa la librería nUML para generar el modelo UML a partir de la información indexada en el apartado anterior. El modelo UML se representar gracias al uso del estándar "XML metadata interchange" que permite recoger toda la información del modelo en un fichero "xmi". Se detallara cada una de las funciones nUML utilizadas y en qué ocasiones ha sido necesaria su participación.

Una vez tenemos un pre-modelo, que no es más que un contendor con toda la información indexada, es necesario transforma esta información en un modelo UML y para ello es imprescindible el uso de la librería nUML, que permite generar un fichero "xmi" para almacenar las características del modelo.

Para describir y explicar cómo se ha utilizado este componente se va a dividir este punto en dos más pequeños.

En el primero se listara la librería nUML y los espacios de nombres que la forman. En el segundo se explicara cómo se ha utilizado dicha librería en este proyecto

Componentes nUML

El uso de nUML se basa en distintos espacios de nombres separados por su utilidad, lo mejor para su comprensión es ir explicándola una a una:

nUML.UML2

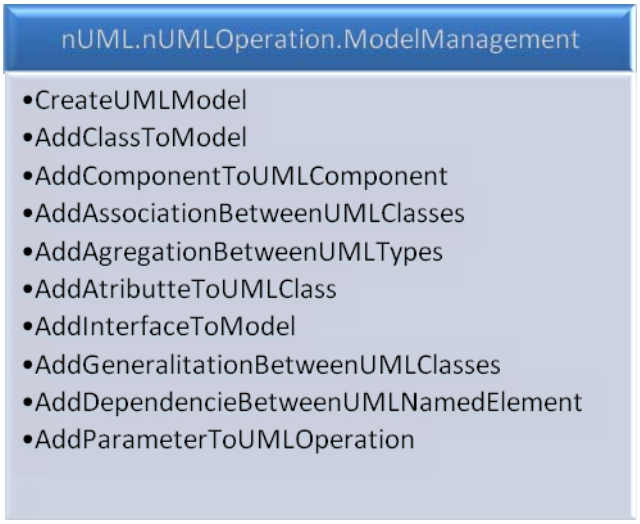
Este espacio de nombres contiene las clases básicas que componen UML 2.0. A continuación se presentan algunas de clases que este componente contiene:

- Class
- Operation
- Action
- Activity
- Actor
- RelationShip
- Comment
- Collaboration
- Call
- Model

nUML.nUMLOperation

Este espacio de nombre contiene todas las operaciones necesarias para la creación de los distintos objetos de nUML. Dentro de este componente la clase que va a ser más utilizado en este proyecto será *nUML.nUMLOperation.ModelManagement* ya que esta clase contiene todas las operaciones estáticas que permiten indexar la información de las soluciones.

A continuación se pasa a listar algunas de las operaciones que se incluyen dentro de la fase:



Uso de la librería nUML

Ahora se expondrá la forma en la que se utiliza la librería nUML para reflejar los elementos de código que anteriormente se han enumerado, en el modelo UML correspondiente.

Como punto de partida indicar que el componente utiliza la interfaz ***Transformer*** por ello se implementan los siguientes métodos:

- ***GetTrasnformLevel:*** Indica si el archivo pasado por parámetro puede ser analizado o no.
- ***ReleaseApplications:*** Se liberan todo los recursos utilizados por el componente.
- ***TrasnformalInternal:*** Método principal, es el encargado de analizar, obtener y devolver toda la información de las soluciones pasadas por parámetros. El modelo obtenido es en nUML.

Se identificará a continuación una a una las operaciones nUML que se utilizan para la obtención de los modelos nUML.

Para crear el modelo se utiliza la operación: *createUMLModel (nombre).*

- **Nombre:** Como nombre de modelo se introduce el nombre de la solución ya que como se comento anteriormente se ha equiparado el concepto modelo con el de solución.

Clases

Para añadir una clase se utiliza: *addClassToUMLModel (nombre, modelo).*

- **Nombre:** Nombre de la clase.
- **Modelo:** El modelo nUML al que se va a insertar la clase.
- Dentro de la clase se definirán las siguientes propiedades.
- **Visibility:** Grado de visibilidad de la clase,
- **IsAbstract:** Si la clase es abstracta

Atributos

Para definir un atributo se emplea: *addAttributetoUMLClass (nombreAtrib, clase).*

- **NombreAtrib:** El nombre del atributo.
- **Clase:** Clase nUML en la que se define el atributo.
- Para cada atributo se define su atributo **visibility**.

Metodos

Para incluir un método dentro de una clase se utiliza la función *addOperationtoUMLClass(nombremetodo, parametrosmetodo, clasedeparametro, tipo de parámetro, clasenUML)*.

- **NombreMetodo:** El nombre del método.
- **Parámetros:** Array con los parámetros del método.
- **TipoParametro:** Array con los tipos de los parámetros.

Una vez definidos los elementos estructurales de la clase definimos las relaciones que esa clase tiene.

Dependencias

Para añadir una dependencia se usa addDependecyBetweenUMLNamedElement (modelo, elemento1, elemento2).

- **Modelo:** modelo nUML.
- **Elemento1:** Se le pasa la clase que en ese momento estamos analizando.
- **Elemento2:** Clase con la que tiene dependencia. Si la clase todavía no existía en el modelo la insertamos.

Asociaciones

Para insertar una asociación se usa *addAssociationBetweenUMLClasses* (clase1, rolclse1, clase2, rolClase2, modelo, nombreAsociacion).

- **Clase1:** La clase que actualmente estamos analizando.
- **rolClase1:** Se le pasa string.emtpy.
- **Clase2:** La clase con la que se hace la asociación.
- **rolClase2:** Se le pasa string.emty.
- **Modelo:** el modelo nUML.
- **NombreAsociacion:** Se le pasa string.emtpy

Agregaciones

Para incluir una agregación se usa *addAgreggationBetweenUMLTypes (clasenUML1, rol1, claseUML2, rol2, modelo, nombre).*

- **clasenUML1:** La clase que actualmente estamos analizando.
- **rol1:** Se le pasa string.empty.
- **claseUML2:** La clase con la que se hace la agregación.
- **rol2:** Se le pasa string.empty.
- **Modelo:** el modelo nUML.
- **Nombre:** string.empty.

Generalizaciones

Para crear una generalización se usa *addGeneralizationBetweenUMLClases (modelo, clasepadre, subclase).*

- **Modelo:** modelo nUML.
- **ClasePadre:** La clase de la que hereda
- **Subclase:** La clase que estamos analizando.

Realizaciones

Para definir un interfaz implementado por la clase se usa *addInterfaceRealizationtoClass(clase, interfaz).*

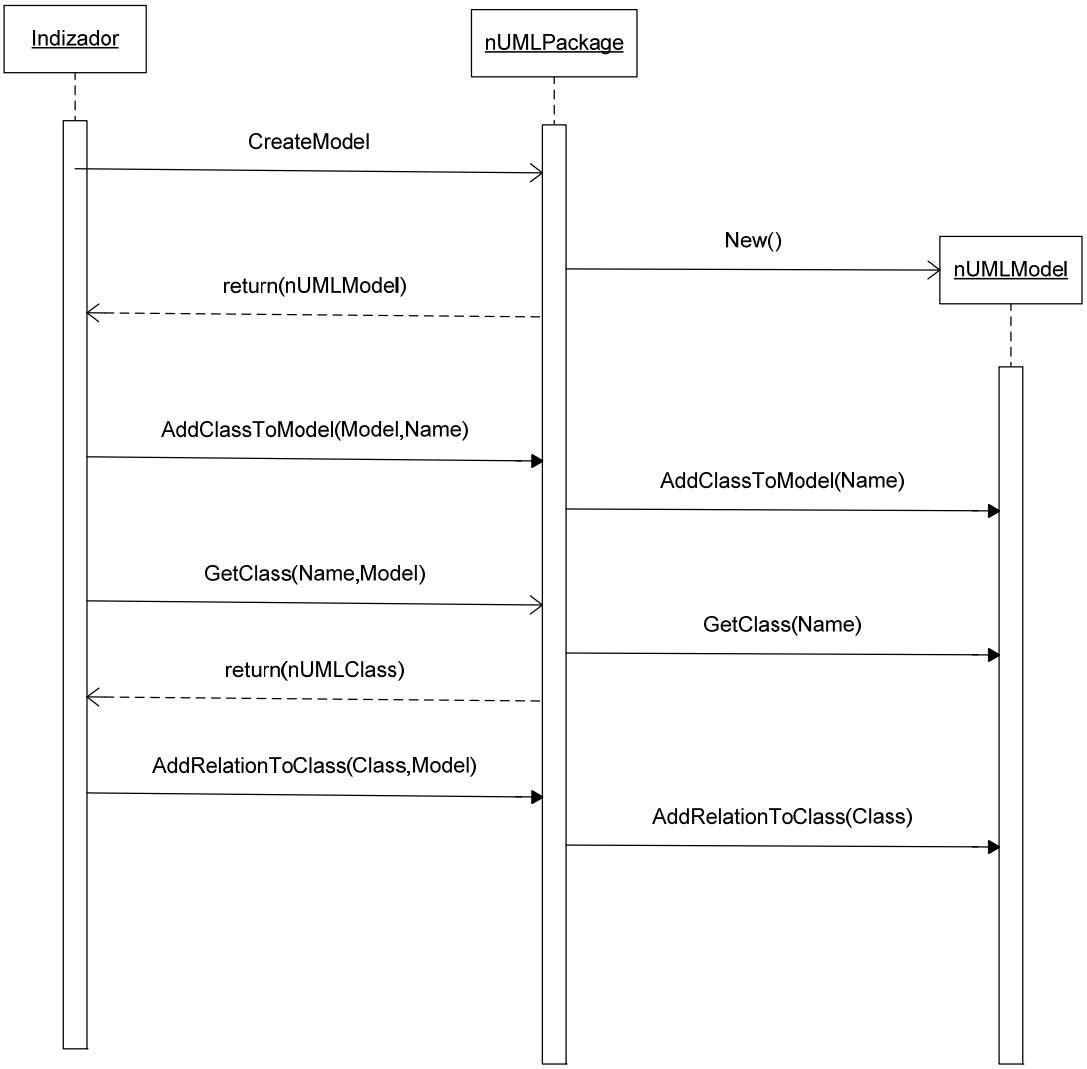
- **Clase:** Clase nUML.
- **Interfaz:** Interfaz nUML.

Interfaces

Para definir las interfaces se usa *addInterfacetoUMLModel (nombre, modelo).*

- **Nombre:** nombre de la interfaz.
- **Modelo:** modelo nUML.
- Definimos la **visibility** de la interfaz.
- Para definir los métodos de la interfaz usamos **addOperationToUMLInterface**.

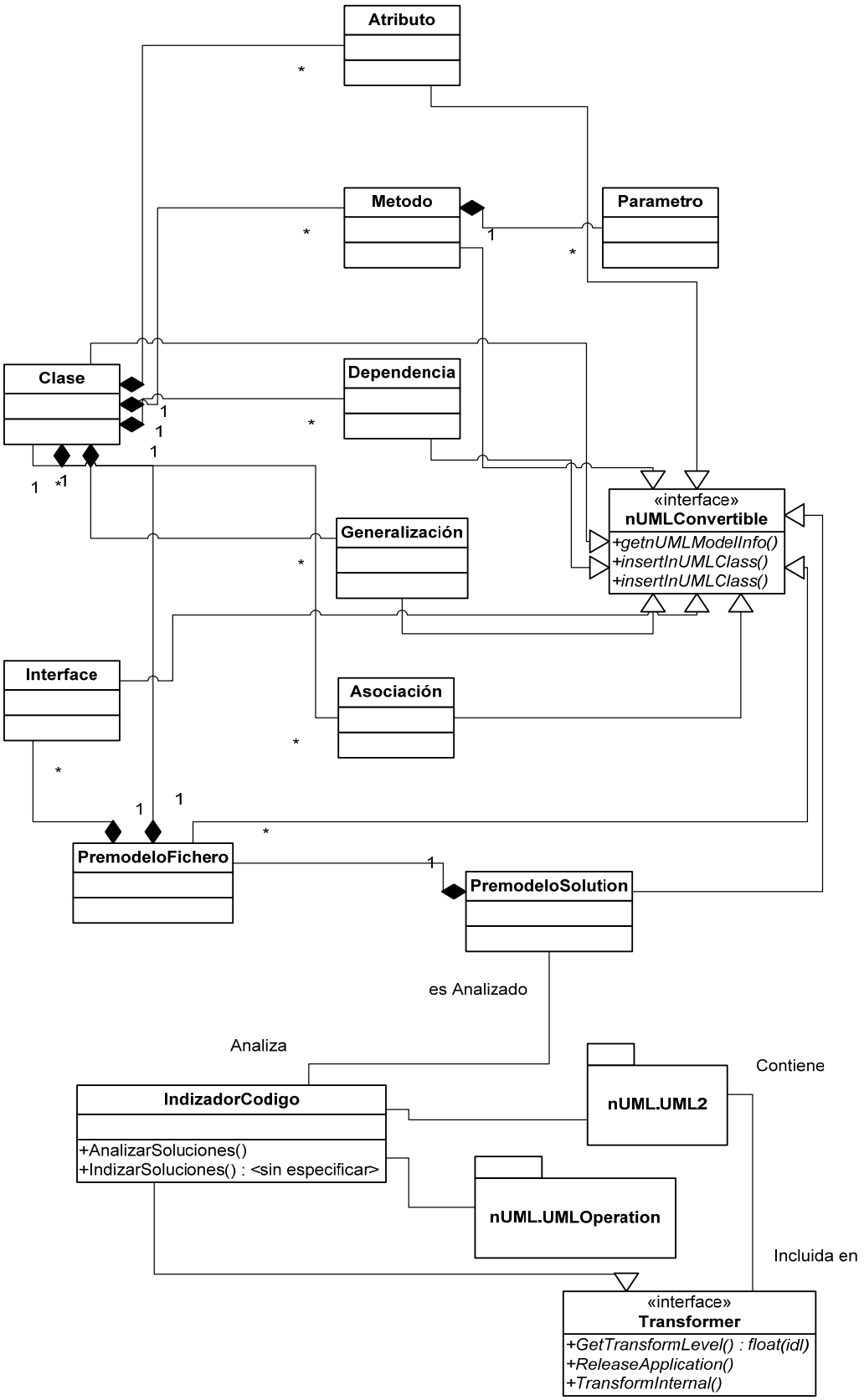
El funcionamiento básico para estas operaciones viene definido según este diagrama:



Como resultado final se obtiene un fichero “xmi” que contiene toda la información asociada la solución indexada. El fichero “xmi” pude ser abierto por muchas herramientas de modelado lo que permite que la información extraída de la solución sea fácilmente distribuible.

Diagrama de Clases

El diagrama de clases resultante para el sistema es el siguiente.


Atributo
Metodo
Parametro
Clase
Dependencia
«interface»
nUMLConvertible
+getnUMLModelInfo()
+insertInUMLClass()
+insertInUMLClass()
Generalización
Interface
Asociación
PremodeloFichero
PremodeloSolution
es Analizado
Analiza
Contiene
IndizadorCodigo
+AnalizarSoluciones()
+IndizarSoluciones() : <sin especificar>
nUML.UML2
nUML.UMLOperation
Incluida en
«interface»
Transformer
+GetTransformLevel() : float(idl)
+ReleaseApplication()
+TransformInternal()

Distribución de plazos

Aunque la realización de este proyecto comenzó hace tres años en el 2007, fue abandonada y retomada de nuevo en abril del 2010. Por ello para la realización de este apartado se ha tomado como referencia esta fecha ya que han sido estos últimos meses los que han supuesto un mayor esfuerzo.

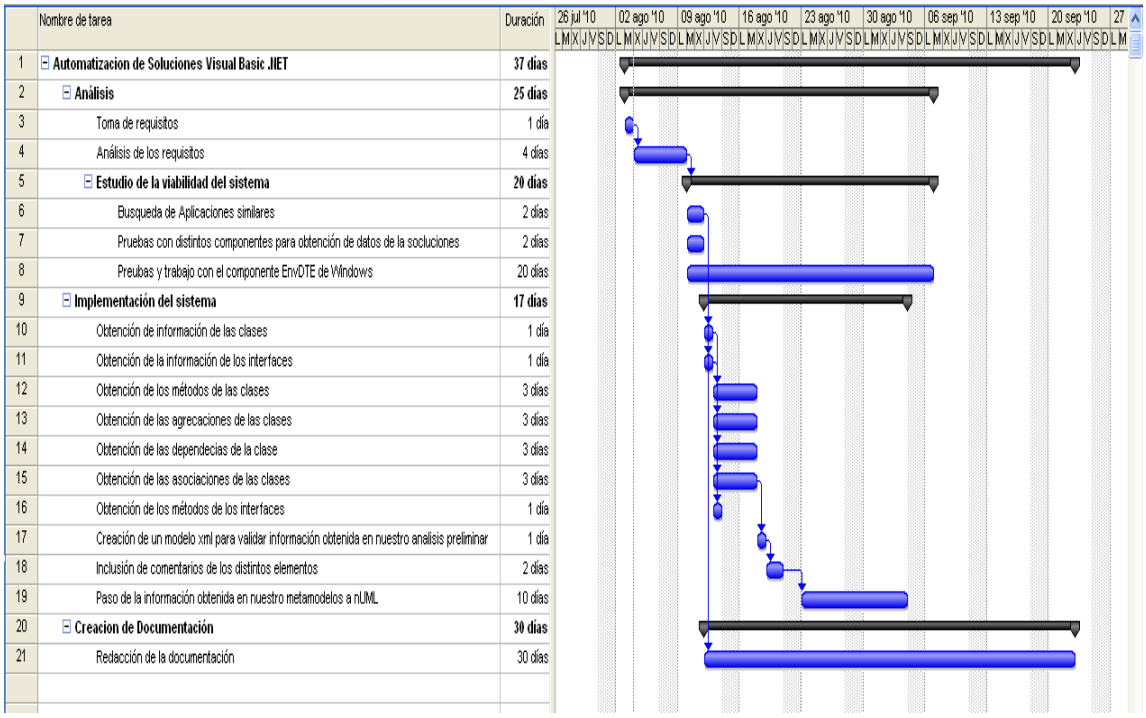
De esta forma la construcción del Indexador de soluciones solicitado comenzó el día 15 de Abril del 2010, y finaliza el 25 de Septiembre del 2010.

Debido a que los desarrolladores de este proyecto compatibilizan este desarrollo con una vida laboral, se interrumpió aproximadamente 15 días en Agosto. Además de ello, debido a que toda la comunicación con el cliente se realizo por medio de email, el delay en puntos clave provoco un retraso de otro mes.

Por lo tanto el esfuerzo empleado ha sido:

- De 15 de Abril de 2010 a 10 de Agosto de 2010, y del 23 de Agosto de 2010 al 20 de Septiembre del 2010.
- Jornada laboral de 4 horas.
- 5 días laborales a la semana, festivos incluidos.

Dentro de este control se incluye el desarrollo de la documentación necesaria.



	Nombre de tarea	Duración
1	**Automatizacion de Soluciones Visual Basic .NET**	**37 días**
2	**Análisis**	**25 días**
3	Toma de requisitos	1 día
4	Análisis de los requisitos	4 días
5	**Estudio de la viabilidad del sistema**	**20 días**
6	Busqueda de Aplicaciones similares	2 días
7	Pruebas con distintos componentes para obtención de datos de la socluciones	2 días
8	Preubas y trabajo con el componente EnvDTE de Windows	20 días
9	**Implementación del sistema**	**17 días**
10	Obtención de información de las clases	1 día
11	Obtención de la información de los interfaces	1 día
12	Obtención de los métodos de las clases	3 días
13	Obtención de las agrecaciones de las clases	3 días
14	Obtención de las dependecias de la clase	3 días
15	Obtención de las asociaciones de las clases	3 días
16	Obtención de los métodos de los interfaces	1 día
17	Creación de un modelo xml para validar información obtenida en nuestro analisis preliminar	1 día
18	Inclusión de comentarios de los distintos elementos	2 días
19	Paso de la información obtenida en nuestro metamodelos a nUML	10 días
20	**Creacion de Documentación**	**30 días**
21	Redacción de la documentación	30 días

El personal involucrado ha sido:

- Dos analistas programadores.

- Un director de proyecto.
- Un profesional en el campo de reuso.

A continuación se ofrece el resumen de todas las horas invertidas en el presente proyecto:

Fase	Nº total de días	Nº total de horas
Desarrollo del sistema	74	296
Redacción de la memoria	25	100
Reuniones del equipo de desarrollo	10	40

Presupuesto

Es necesario incluir además el costo de las horas y de los materiales utilizados para el desarrollo del proyecto.

Recursos Humanos

Dentro del desarrollo se encuentra personal con distintas características y conocimientos y debido a ello con distintos honorarios lo que hace necesario la inclusión de las tarifas de los mismos.

Perfil en informática	Características	Salario Bruto Anual
Analista Programador	Experiencia entre 2 y 5 años	24000
Director de Proyecto	Experiencia de más de 10 años	40000

Teniendo en cuenta que el número de horas por convenio son aproximadamente 1810 para el sector de técnicos el coste hora quedaría:

Perfil en informática	Características	Coste Hora
Analista Programador	Experiencia entre 2 y 5 años	14
Director de Proyecto	Experiencia de más de 10 años	22

Según la implicación que ha tenido cada uno y a partir de la información recogida en el punto anterior, es posible calcular el coste total en recursos humanos.

Recursos Hardware

A continuación se detalla la inversión realizada en los componentes hardware utilizados durante todo el proceso de desarrollo.

Componente	Coste(€)
Ordenador portátil	**1000,00**
TOTAL (con IVA)	**1000,00**

Recursos Software

Para este punto se tienen en cuenta las licencias de las aplicaciones empleadas en el proceso de desarrollo del componente.

Recurso	Coste (€/licencia)
Sistema operativo Windows XP Professional	194 €
Entorno de desarrollo Microsoft Visual Studio .NET 2008	667€
Microsoft Office 2007	314€
Microsoft Visio 2007	667€
Microsoft Project 2007	313€
TOTAL (con IVA)	**2155€**

Fungibles

Recurso	Unidades	Coste/unidad	Coste/recurso
Paquete de folios (500 folios)	1	6€	6€
Almacenamiento: Memoria usb	1	25€	25€
Cuaderno hojas cuadriculadas	2	2€	4€
Bolígrafos	5	0,50€	2,5€
Impresión Documentación	4	12€	48€
	TOTAL (con IVA)		**85,5€**

Formación

Debido a la naturaleza del proyecto, se considera que el coste de la formación de los miembros del equipo de desarrollo queda incluido en los anteriores apartados.

Resumen de costes

Concepto	Coste
Recursos humanos	
Hardware	1000
Software	2155
Fungibles	85,5
TOTAL (con IVA)	**3330,5**

Resultados conclusiones y desarrollos futuros

Resultados

Durante la realización de este proyecto se ha abordado la creación de una aplicación que permitiera extraer información de soluciones de Visual Studio que contuvieran proyectos de lenguaje C# válidos, con el objetivo de poder utilizar esta información dentro de un contexto de reutilización de software. Este indexador tenía como premisa devolver toda la información analizada en UML, lenguaje de modelado de aplicaciones que permite recoger las características y el funcionamiento de un sistema, de manera que fuera más fácil su posterior tratamiento con vistas a una futura reutilización.

Una característica muy importante a tener en cuenta en este proyecto ha sido que no se trataba de un sistema aislado, sino de una pequeña parte de un sistema mucho mayor que tiene como objetivo generar una gran base de conocimiento, utilizando fuentes totalmente heterogéneas. Por ello este proyecto es uno más entre una gran cantidad de indexadores que obtienen la información de una gran variedad de orígenes. Por consiguiente la aplicación puede verse como una pequeña "pata" de un gran sistema matriz de reutilización de conocimiento que obtiene información heterogénea de distintos indexadores y genera una salida homogeneizada en UML.

Para la consecución de este proyecto ha sido necesario el estudio y posterior utilización de tres tecnologías fundamentalmente, que son las que intervienen en los procesos de ejecución clave del sistema.

- **Ioniz.zip:** Librería de descompresión de ficheros .zip.
- **EnvDTE y EnvDTE80:** Bibliotecas COM que permiten la automatización de Visual Studio.
- **nUML:** Librería que permite transformar la información indexada en UML.

Gracias a la combinación de estas tres tecnologías hemos conseguido desarrollar un sistema que es capaz de leer como entrada un fichero comprimido con formato .zip, descomprimir este fichero para acceder a las soluciones que contenga, indexar la información encontrada en el código C# de estas soluciones y transformar toda esta información en UML. De esta forma se puede afirmar que hemos conseguido cumplir con todos los objetivos planteados inicialmente.

Conclusiones

Dada la naturaleza de este proyecto el principal problema que ha sido necesario afrontar fue conseguir un conocimiento suficientemente amplio de todas las tecnologías implicadas para poder realizar un adecuado uso de las mismas, en pos de conseguir los objetivos marcados. En la mayoría de los proyecto las tecnologías a aplicar son conocidas de antemano, de esta forma el trabajo se reduce a encontrar la mejor solución al problema planteado. Además este conocimiento a priori de la tecnología, permite realizar una estimación de costes (tanto en tiempo como en recursos materiales) más completa y precisa. En este proyecto la situación ha sido ligeramente diferente ya que únicamente se conocía el entorno de desarrollo que se debía utilizar, .Net. Se ha dado por lo tanto el caso de tener que utilizar tecnologías que eran totalmente desconocidas como la Liberia nUML, o incluso tener que realizar una amplia labor de investigación para determinar que tecnología era necesario aplicar, como en el caso de las bibliotecas COM EnvDTE y EnvDTE80. Esto ha provocado que en términos absolutos se haya invertido mucho más tiempo en "romper" las tecnologías, es decir conseguir un conocimiento adecuado de las mismas, que en aplicarlas posteriormente en la consecución del proyecto. De gran ayuda ha sido la documentación facilitada, tanto la aportada por el grupo de "Knowledge Reuse" para el uso de nUML, como la documentación en línea de Microsoft.

Por otro lado la realización de este proyecto ha sido el fruto del esfuerzo de más de una persona, por lo que el trabajo en equipo ha sido clave para conseguir todos los objetivos marcados, y aunque ha sido necesario un esfuerzo extra para que la colaboración no degenerara en un problema, los beneficios obtenidos en comparación han sido enormes.

A título personal este proyecto me ha permitido aplicar muchos conocimientos obtenidos a lo largo de la carrera y sobre todo recordar muchos de ellos, por desgracia olvidados dentro del mundo profesional que al menos yo he conocido, y en el que muchas metodologías y patrones de actuación más que recomendables son totalmente ignorados. No es necesario mencionar que la realización de este proyecto me ha posibilitado conocer un poco mejor el campo de la reutilización de conocimiento y generarme una idea de la situación actual y de cual podrían ser futuras líneas de actuación.

Desarrollos Futuros

Dentro del marco de este proyecto se pueden identificar claramente dos puntos que permiten un desarrollo futuro que no implica una inversión en costes muy elevada:

- Aumentar el conjunto de tecnologías de compresión que el indexador es capaz de descomprimir.
- Ampliar el conjunto de lenguajes .net a indexar.

Aumentar las tecnologías de compresión

Aunque .zip es el estándar de compresión con mayor difusión existen otros como: rar, 7z o tar.gz que también se encuentran muy extendidos. De esta forma si el indexador fuera capaz de recibir como entrada ficheros comprimidos en estas tecnologías se podría analizar una mayor cantidad de código, lo que permitiría generar por lo tanto un base de conocimiento mucho mayor.

Ampliar el conjunto de lenguajes .net a indexar

En la versión actual del indexador los lenguajes .net que es capaz de analizar son: C# y VB.net. Sin embargo debido a que el análisis de los elementos de código y relaciones existentes en los ficheros fuentes se hace a través de las interfaces EnvDTE y EnvDTE80, cualquier otro lenguaje .net es susceptible a ser indexado con esta aplicación sin tener que realizar ningún cambio considerable, es más únicamente sería necesario cambiar la comprobación que realizamos sobre la extensión de los ficheros de código e incluir, junto a ".cs" y ".vb", la extensión del lenguaje .net deseado.

Bibliografía

Durante la realización de este proyecto ha sido internet la fuente de información a la que se ha recurrido en la gran mayoría de las veces, por ello la bibliografía de este proyecto está compuesta en su mayoría por referencias web.

- http://msdn.microsoft.com
- http://www.uml.org/
- http://es.wikibooks.org/wiki/C_sharp_NET
- http://www.canalvisualbasic.net/manual-net/c-sharp/
- http://es.wikipedia.org/wiki/Lenguaje_Unificado_de_Modelado
- http://www.dcc.uchile.cl/~psalinas/uml/introduccion.html
- http://office.microsoft.com/es-es/visio-help/CH010064890.aspx
- http://www.codeproject.com/
- El lenguaje de modelado unificado. Autores: Grady Booch, James Rumbaugh e Ivar Jacobson. Editorial: Addison Wesley.
- http://blogs.msdn.com/b/dotnetinterop/archive/2006/04/05/.net-system.io.compression-and-zip-files.aspx
- http://www.canalvisualbasic.net/manual-net/c-sharp/#baseClass